VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Editor pro podporu tvorby Storyboardů

Storyboard Editor

2012 Bc. Marek Garbulinský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Marek Garbulinský**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Editor pro podporu tvorby Storyboardů
Storyboard Editor

Zásady pro vypracování:

Cílem je vytvořit program, který usnadní vytváření Storybardů, což je vlastně grafický popis činnosti procesu. Programu umožní vytvářet jednoduché malé ikony (části) ze kterých se následně poskládá celý obrázek (záběr). Tyto záběry se pak poskládají do jedné nebo více sekvencí, které reprezentují popis celého procesu. Důležitou vlastností vytvářeného programu bude definování aktivních oblastí v jednotlivých ikonách a vazeb mezi ikonami prostřednictvím těchto vlastností.

Editor bude umožňovat:
1. Vytváření ikon a jejich popisu (samotný obrázek bude importován ve formátu SVG).
2. Vytváření aktivních oblastí v jednotlivých ikonách a popisu příslušných vazeb, které mohou být realizovány pomocí této aktivní oblasti.
3. Definici vazeb mezi jednotlivými ikonami (dědičnost, propojení, agregace).
4. Skládání Storyboardů a jednotlivých záběrů z předefinovaných ikon.
5. Export vytvořených Storyboardů.

Seznam doporučené odborné literatury:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
JŮCHOVÁ, V.; ŠTOLFA, S.; JEŽEK, D.; VONDRÁK, I. Storyboards in Business Process Modeling. In 8th Industrial Simulation Conference 2010, LENCSE, G.; LÁSZLO, M. eds., Budapest, Hungary: Eurosis 2010. pp. 57-61. ISBN 978-90-77381-55-7.
Dále podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 4. května 2012 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012 .....

Děkuji Ing. Davidu Ježkovi, Ph.D. za odborné metodické vedení a cenné rady při zpracování mé práce.

Abstrakt

Cílem této diplomové práce je seznámit čtenáře s alternativní metodou pro modelování podnikových procesů. V první části se věnuji metodě storyboardů, která poskytuje jiný pohled na modelování podnikových procesů. Druhá část práce popisuje vývoj editoru pro vytváření storyboardů. Na začátku této části jsem definoval požadavky na daný systém. Poté se věnuji návrhu architektury systému. Závěr této práce je věnován samotné implementaci. Pro vývoj aplikace byla zvolena platforma Java.

Klíčová slova: Editor, Storyboard, Java, Swing, Podnikové procesy

Abstract

The aim of the thesis is to familiarize the reader with an alternative method of business process modeling. The fist part deals with a method of storyboards which provides a different view of business process modeling. The second part describes a development of Storyboard Editor. At the beginning of this part, requirements for the system are described. Based on the aforementioned requirements, an architecture of the system is designed. The final part of the thesis deals with the implementation itself. The Java platform was chosen for the application development.

Keywords: Editor, Storyboard, Java, Swing, Business Process

Seznam použitých zkratek a symbolů

BPR	–	Business Process Reengineering
IDEF	–	Integration DEFinition
EPC	–	Event-driven Process Chains
UML	–	Unified Modeling Language
BPMN	–	Business Process Model and Notation
SVG	–	Scalable Vector Graphics
XML	–	Extensible Markup Language
GUI	–	Graphical user interface
SVN	–	Subversion
FEAT	–	Features
REQ	–	Request
MVC	–	Model-View-Cotroller
UUID	–	Universally unique identifier
DND	–	Drag and Drop

Obsah

1 Úvod **6**

2 Úvod do Storyboardů **7**
2.1 Proces . . . 7
2.2 Procesní řízení . . . 7
2.3 Podnikový proces / Business process . . . 7
2.4 Modelování podnikových procesu . . . 8
2.5 Business Process Reengineering . . . 8
2.6 Model podnikového procesu . . . 9
2.7 Metody pro modelování podnikových procesů . . . 9
2.8 Problém při simulaci . . . 9

3 Storyboard **11**
3.1 Historie Storyboardů . . . 11
3.2 Snímek . . . 11
3.3 Ilustrační obrázek / Shot . . . 11
3.3.1 Artefakty / ShotPart . . . 11
3.4 Aktivní oblast /RelationshipArea . . . 13
3.5 Činnost aktivní oblasti / RelalitionSemantic . . . 13
3.6 Příklad . . . 13
3.7 Zachycení metody Storyboardů pomocí třídního diagramu . . . 14
3.8 Závěr do úvodu Storyboardů . . . 15

4 Analýza **16**
4.1 Specifikace požadavků . . . 16
4.1.1 Knihovna ikon . . . 16
4.1.2 Aktivní oblasti . . . 16
4.1.3 Činnost . . . 16
4.1.4 Skládání Storyboardů . . . 17
4.1.5 Export . . . 17
4.2 Dokumentace požadavků . . . 17
4.3 Použité technologie . . . 17
4.3.1 Java 7 . . . 17
4.3.2 Batik . . . 18
4.3.3 MigLayout . . . 19
4.3.4 Netbeans . . . 19
4.3.5 SVN . . . 19

5 Návrh **20**
5.1 Architektura - složení projektu . . . 20
5.1.1 Rozhraní PrvekProjektu . . . 20
5.1.2 Třída AbstraktniPrvekProjektu . . . 20

5.1.3 Rozhraní - Film, Storyboard, Proces, Snimek . . . 21
5.1.4 Třídy - FilmImpl, StoryboardImpl, ProcesImpl, SnimekImpl . . . 21
5.1.5 Rozhraní Snímek . . . 22
5.1.6 Třída Platno . . . 22
5.1.7 Rozhraní GrafickýObjektNaPlatne . . . 22
5.1.8 Rozhraní IAktivniOblastPrijmajici . . . 22
5.1.9 Návrhový vzor Visitor . . . 23
5.1.10 Použití Visitoru - export do xml . . . 24
5.2 Aplikace pro knihovnu ikon . . . 25
5.2.1 Serializace Ikony . . . 25
5.2.2 Třída Činnost . . . 27
5.2.3 Třída PovolenaIkona . . . 27
5.2.4 Třída NakreslenaIkona . . . 27
5.2.5 Třída NakreslenaSvgIkona . . . 27
5.2.6 Třída IkonaUpravitelneOblasti . . . 28
5.2.7 Třída AktivniOblastVychoziIkony . . . 28
5.2.8 Třída KnihovnaIkon . . . 28
5.2.9 Drag and drop . . . 29
5.2.10 Správa knihovny ikon . . . 29
5.3 StoryBoardEditor . . . 31
5.3.1 Modul Detail . . . 32
5.3.2 Modul Projekt . . . 34
5.3.3 Modul Knihovna ikon . . . 36
5.3.4 Modul Menu . . . 36
5.3.5 Modul Frame . . . 39
5.4 Shrnutí vývoje aplikace pro podporu metody Storyboardů . . . 40

6 Závěr **42**

7 Reference **43**

Přílohy **43**

A Příloha tabulky **44**

B Příloha Zdrojové kódy **45**

C Uživatelská příručka **48**
C.1 Instalace a spuštění programu Storyboardbuilder . . . 48
C.2 Vytvoření nového projektu . . . 48
C.2.1 Zvolení knihovny ikon . . . 49

D Obsah přiloženého CD **51**

Seznam tabulek

1 Metody pro byznys modelování 9
2 FEAT 1 44
3 FEAT 2 44
4 FEAT 4 44
5 FEAT 5 44

Seznam obrázků

1 Podnikový proces (převzato z [2]) . . . 7
2 Objednání produktu . . . 9
3 Vstupní artefakty procesu (převzato z [5]) . . . 11
4 Role (převzato z [5]) . . . 12
5 Kontext procesu (převzato z [5]) . . . 12
6 Aktivní oblasti u artefaktů (převzato z [5]) . . . 13
7 Ilustrace kompletního snímku (převzato z [5]) . . . 14
8 Třídní diagram metody Storyboardů (převzato z [5]) . . . 14
9 Analytický model projektu . . . 20
10 Architektura projektu . . . 21
11 Ukázka strukury projektu v aplikaci. . . . 22
12 Detail snímku . . . 23
13 Implementace návrhového vzoru Visitor . . . 24
14 Editace knihovny ikon . . . 25
15 Náhled na knihovnu ikon a její ikony . . . 29
16 Storyboard editor . . . 31
17 Náhled na detail ikony . . . 32
18 Balíčky modulu detailu . . . 32
19 Struktura balíčku GUI . . . 33
20 Struktura balíčku model . . . 33
21 Struktura balíčku service . . . 34
22 Struktura balíčku GUI projekt . . . 34
23 Projekt pomocí JTree . . . 34
24 Zobrazení knihovny ikon . . . 36
25 Menu aplikace . . . 37
26 Struktura balíčku menu . . . 37
27 Nový projekt . . . 37
28 Třídní diagram pro dialogové okno - nový projekt . . . 38
29 Snímek v záložce . . . 40
30 Sekvence snímku v procesu . . . 40
31 Nový projekt . . . 48
32 Nový projekt . . . 48
33 Zvolení umístění projektu . . . 49
34 Připojení ikon . . . 49
35 Výběr ikon . . . 50
36 Potvrzení vytvoření projektu . . . 50

Seznam výpisů zdrojového kódu

1 Operátor diamant 18
2 Výpis programu: String ve výrazu switch 18
3 Výpis programu: Použití komponenty JSVGCanvas 18
4 Výpis programu: implementace visitoru ve třídě Snímek 23
5 Výpis programu: použití visitoru - export do xml 24
6 Výpis programu: Serializační proxy 26
7 Výpis programu: Sestavení projektu pomocí návrhového vzoru visitor . . 35
8 Výpis programu: Spuštění visitoru 35
9 Výpis programu: Konfigurace hlavního okna 39
10 Výpis programu: Implementace visitoru 45
11 Výpis programu: Implementace visitoru 46

1 Úvod

Ze statistického odhadu [1] vyplývá, že 60 procent firem využívá procesní řízení společnosti, na které přechází z funkčního řízení. V dnešní době se jeví procesní řízení jako nejlepší varianta k řízení podniku. Procesní řízení dokáže popisovat jednotlivé procesy modelem. Tento model lze pak následně různě simulovat, upravovat, zpřesňovat, a tím dokáže procesy efektivně řídit. Tento přechod má bohužel svou daň, modelovat podnikové procesy není jednoduchou záležitosti. Firmy jsou nuceny platit byznys analytiky, kteří pomocí různých metodik zachycují podnikové procesy. Někdy používají striktně formální metody, jindy neformální metody. Všechny zainteresované osoby, které se zajímají o podnikové procesy, musí chápat tyto metodiky, které využívají byznys analytici. Pochopení těchto metodik je náročné na čas, ale i finance (zaškolení zaměstnanců). Úvodu do podnikových procesů a jejich modelaci bude vyhrazena první kapitola. Druhá kapitola se zaměří na alternativní metodu modelování podnikových procesů. Alternativou se ukázala metoda Storyboardů, která popisuje procesy pomocí jednoduchých obrázků. Každý proces je zachycen sadou obrázků znázorňující proces. Metoda Storyboardů také zachycuje jistou formalitu, kterou lze využít pro simulaci podnikových procesů. Druhá část této diplomové práce se zabývá návrhem a implementací podpůrné aplikace pro vytváření Storyboardů. Tato kapitola provede čtenáře přes počáteční analýzu požadavků na aplikaci k samotné implementaci.

[1] Procesní řízení - jak si stojí firmy v CŘ? - http://bpr.panrepa.org/Jak_si_stoji.pdf

2 Úvod do Storyboardů

V dnešní době řízení společností není jednoduché. A efektivní řízení společnosti je ještě těžší. Společnosti jsou neustále nuceni zlepšovat své podnikové procesy, aby uspokojili přání svých zákazníků. Každý zákazník má jiné přání. Společnost, která se chce v konkurenčním prostředí udržet a zároveň plnit požadavky svých zákazníku, musí být řízena efektivně. Proto většina podniků přechází od funkčního řízení k procesnímu řízení společnosti, které dokáže lépe reagovat na potřebné změny. Blíže představím tyto pojmy:

- Proces
- Procesní řízení
- Podnikový proces / Bussnise proces

2.1 Proces

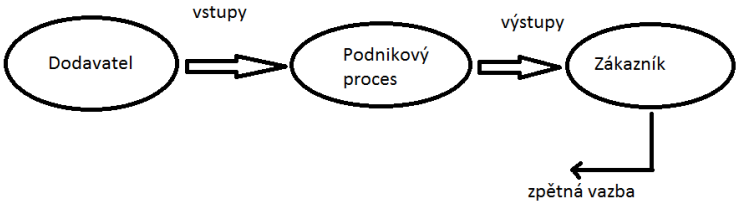
Proces je základním objektem procesního řízení. Definici procesu můžeme formulovat takto: "Proces je soubor vzájemně působících činností, který přeměňuje vstupy na výstupy."[1]

2.2 Procesní řízení

Procesní řízení je nový pohled na organizaci a řízení činností v podnicích, vychází z myšlenky, že každý produkt nebo služba je vytvářena sledem činnosti tedy procesem, a to bez ohledu na organizační uspořádání. Až když je proces nadefinován, jsou nadefinováni pracovníci, kteří jednotlivé činnosti budou provádět. Tyto činnosti lze následně měřit, vyhodnocovat a po případě měnit.

2.3 Podnikový proces / Business process

"Podnikový proces je souhrnem činností transformujících souhrn vstupů do souhrnu výstupů (zboží nebo služeb) pro jiné lidi nebo procesy."[2] Příkladem jednoduchého pod-



Obrázek 1: Podnikový proces (převzato z [2])

nikového procesu, může být vyřízení objednávky. Zákazník si objednává (třeba pomocí internetu) produkt firmy XYZ. Scénář takovéhoto procesu může být následující. Zákazník si vybere produkt a odešle objednávku. Informační systém firmy XYZ zpracuje danou objednávku a uloží do seznamu nevyřízených objednávek. Pracovník firmy XYZ, který je zodpovědný za vyřízení objednávky, se podívá na seznam nevyřízených objednávek a zjistí, že je nová objednávka. Pracovník přes sklad zařídí, aby se připravil produkt pro odeslání. Pracovník ve skladu zabalí daný produkt a odnese balík na expediční oddělení. Zde na expedičním oddělení balík odešlou na adresu zákazník. Takto jsme si nadefinovali jednoduchý podnikový proces.

2.4 Modelování podnikových procesu

Důvodem, proč je zapotřebí modelovat podnikové procesy, je zachycení a porozumění daného podnikového procesu. Modelování podnikových procesů umožňuje namodelovat model podnikového procesu. S tímto namodelovaným podnikovým procesem se můžou seznámit všichni zaměstnanci společnosti a seznámit se, jak daný proces funguje. Model procesu může být prvním podkladem pro návrh informačního systému daného podniku. Model lze také využít na reengineering (redesign) procesů podniků. Model můžeme použít pro řízení a správu procesů. Na základě vzniklých modelů můžeme simulovat podnikové procesy. Výstupy těchto simulací lze využít při optimalizaci podnikových procesů a plánování zdrojů.

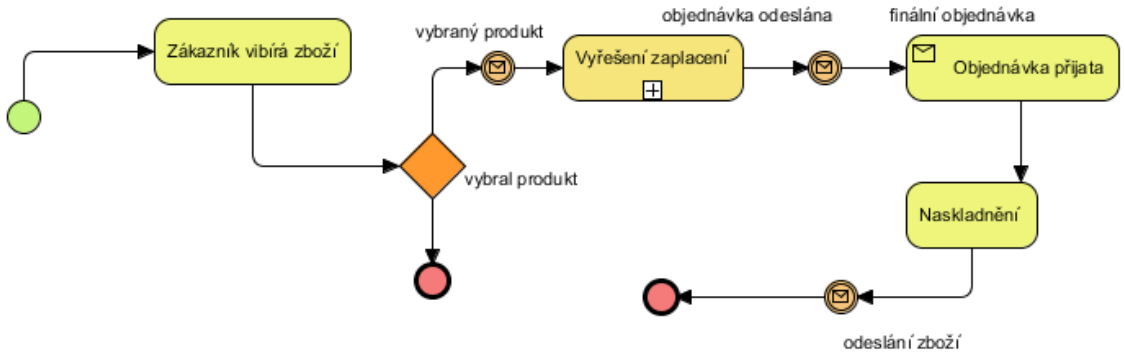
2.5 Business Process Reengineering

Business Process Reengineering (dále jen BPR) je metoda, jak zlepšit podnikové procesy. Tato metoda vychází z myšlenky, že dosavadní proces je zcela nepřípustný. Ve výsledku se celý proces změní od začátku. Model podnikového procesu může sloužit jako základ pro BPR.

2.6 Model podnikového procesu

Model byznys procesu je abstraktní reprezentace byznys procesu obvykle umožňující jeho další zpracování automatizovaným způsobem. Model může být formální nebo neformální. Modelovaní těchto podnikových procesů provádějí především IT specialisté, budˇ to jako interní zaměstnanci, nebo pomocí outsourcingu. Tito IT specialisté využívají různé techniky pro namodelovaní podnikového procesu. Podívejme se na jednoduchý příklad takovéhoto modelu 2.

Příklad 2.1
Objednání produktu pomocí BPMN ■



Obrázek 2: Objednání produktu

2.7 Metody pro modelování podnikových procesů

Pro modelování podnikových procesů existují různé metody/nástroje/metodiky, kterými lze zachytit podnikový proces. Tyto metody mohou být od neformálních typů až po striktně formální. Přehledný seznam metodik můžeme vidět na následující tabulce 1:

Formální metody	Semiformální metody
Konečné automaty	IDEF
Petriho sítě	EPC
Procesní algebra - Pi kallkul	UML
Ontologie	BPMN

Tabulka 1: Metody pro byznys modelování

2.8 Problém při simulaci

Aby bylo možné simulovat podnikový proces, je nutné podrobně popsat daný podnikový proces v jednom z mnoha jazyků vhodných pro simulaci. Bohužel tyto jazyky nejsou příliš

srozumitelné, a proto simulační modely musí vytvářet expert v oblasti simulací. Avšak expertovi pro vytváření simulací zase chybí potřebně znalosti podnikového procesu. Vytvoření simulačního modelu pak musí předcházet mnoho schůzek mezi zaměstnanci společnosti a expertem z oblastí simulací, na kterých vzniká většinou neformální popis procesu, který je až následně ručně převeden do formálního simulačního modelu.

Je zřejmé, že modelování podnikových procesů přináší danému podniku mnoho výhod oproti konkurenci. Bohužel za cenu složitosti takovéto modely modelovat. Cílem této diplomové práce je seznámit uživatele s jiným přístupem k modelaci modelů. Tento přístup využívá metodu Storyboardů. Pro tuto metodu jsem dostal za úkol vytvořit program, který dokáže jednoduše vytvářet Storyboardy.

3 Storyboard

Hlavním cílem metody Storyboardů je snaha o popsání bussnise procesů lidštější formou, které rozumí každá zainteresovaná osoba v procesu. Dalším důležitým cílem je možnost jednodušeji modelovat podnikové procesy. Storyboardy také umožňuji vygenerovat formální popis bussnise procesu, který může být základem pro budoucí simulační model.

3.1 Historie Storyboardů

Základní koncept Storyboardů byl představen v dokumentu *Storyboards in Business Process Modeling*: "Storyboard je multimediální obrázek, který obsahuje sekvenci po sobě jdoucí snímků, každý snímek představuje jeden konkrétní byssnise proces (obchodní proces) Fragment - jedná akce. "[4]

3.2 Snímek

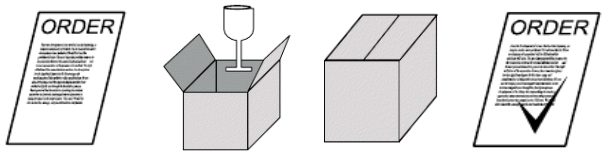
Snímek graficky zobrazuje jeden konkrétní průchod byznys procesem. Snímek obsahuje ilustrační obrázek, seznam vstupních, výstupních artefaktů. Snímek obsahuje také jedinečný název zachycující podnikový proces v dané firmě. Tyto data paky lze využít k poloautomatickému sestavení modelu procesu například pomocí vícedimenzionálních automatů.

3.3 Ilustrační obrázek / Shot

Každý snímek je reprezentován ilustračním obrázkem, tento obrázek se skládá z pozadí snímku a ze vstupních a výstupních artefaktů (grafických ikon). Artefakty jsou v průběhu procesu vytvářeny, používány nebo měněny.

3.3.1 Artefakty / ShotPart

Jako artefakty můžeme považovat dokument, zařízení, soubor atd. Artefakty jsou v průběhu procesu vytvářeny, používány nebo změněny. Podívejme se níže na ilustraci různých artefaktů, které mohou být použity v procesu naskladnění objednávky.

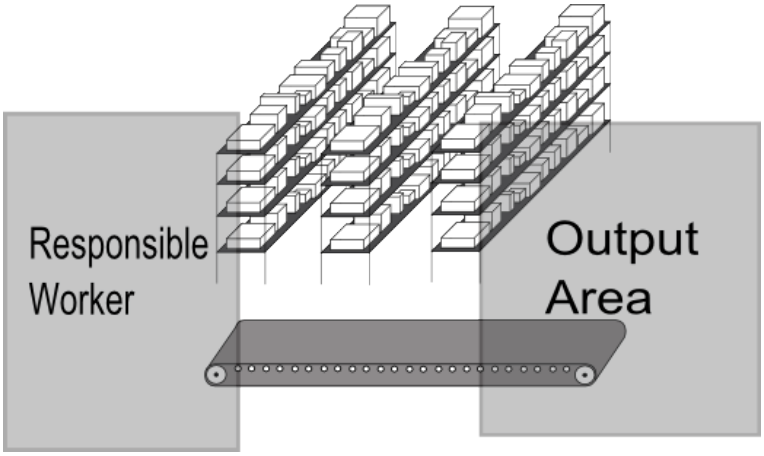


Obrázek 3: Vstupní artefakty procesu (převzato z [5])

3.3.1.1 Role Proces je souhrn činností transformující vstupy na výstupy. Tyto činnosti mohou provádět stroje, lidské zdroje. V našem jednoduchém příkladě bude činnost naskladnění provádět skladník. Je nutné, aby bylo zřejmé, kdo tyto činnosti provádí. Tuto roli opět zachytíme ikonou. Role patří do artefaktů, pouze přidává artefaktu určitou roli, abychom mohli vystihnout právě skladníka v procesu.

Obrázek 4: Role (převzato z [5])

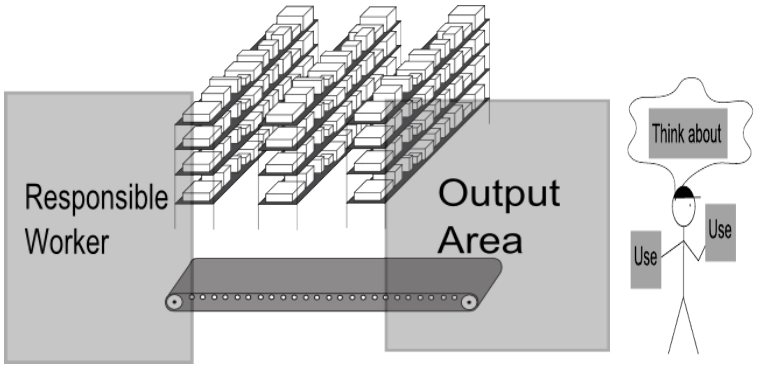
3.3.1.2 Pozadí Celý proces může být zasazen do nějakého kontextu, v našem případě je kontextem sklad. Toto pozadí nám také umožňuje nahlédnout na celkový proces naskladnění objednávky. Pozadí definuje rovněž umístění artefaktů pomocí aktivních oblastí. Celý proces začíná vstupními artefakty, zodpovědnými pracovníky, ze kterého vzniknou výstupy potvrzená objednávka a zabalený produkt. Z takto nadefinovaného pozadí již můžeme získat zajímavé informace z procesu, které mohou být využity pro simulační modely.



Obrázek 5: Kontext procesu (převzato z [5])

3.4 Aktivní oblast /RelationshipArea

Aktivní oblast je rozšíření artefaktů (ikon), které umožňuje nadefinovat vztah mezi artefakty (ikonami). Dále může aktivní oblast uchovávat informace o tom, které ikony lze připojit k dané aktivní oblasti. Pro další zpracování informací ze Storyboardů je získání informací z této aktivní oblasti nejdůležitější, neboť aktivní oblast zachycuje danou činnost, kterou provádí, a také s čím tuto činnost provádí. V našem případě naskladnění zboží můžeme rozšířit ikony skladníka a skladu. Skladník má aktivní oblast ruky, kde může vložit třeba objednávku. Skladu nadefinujeme vstupní aktivní oblast a výstupní aktivní oblast, které budou představovat vstupy a výstupy procesu.



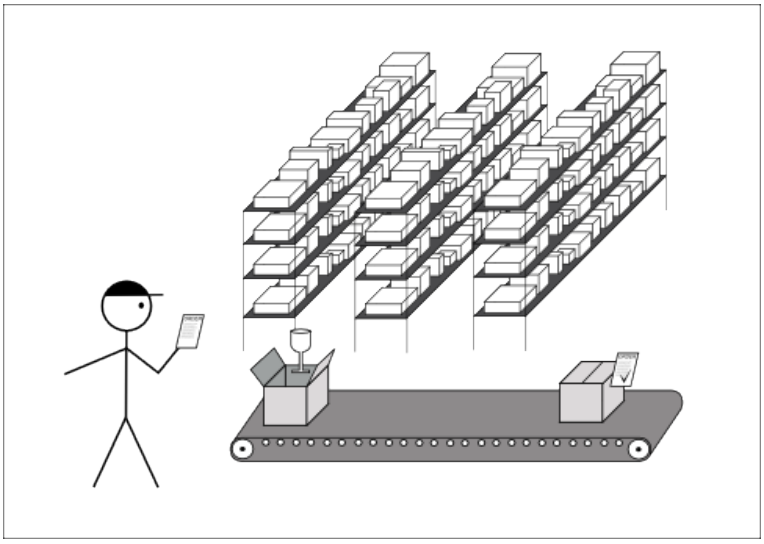
Obrázek 6: Aktivní oblasti u artefaktů (převzato z [5])

3.5 Činnost aktivní oblasti / RelalitionSemantic

Aktivní oblasti můžeme dále obohatit o informaci, jakou činnost lze s touto oblastí provádět. Jako příklad můžeme uvést, skladníka zpracovávajícího objednávku.

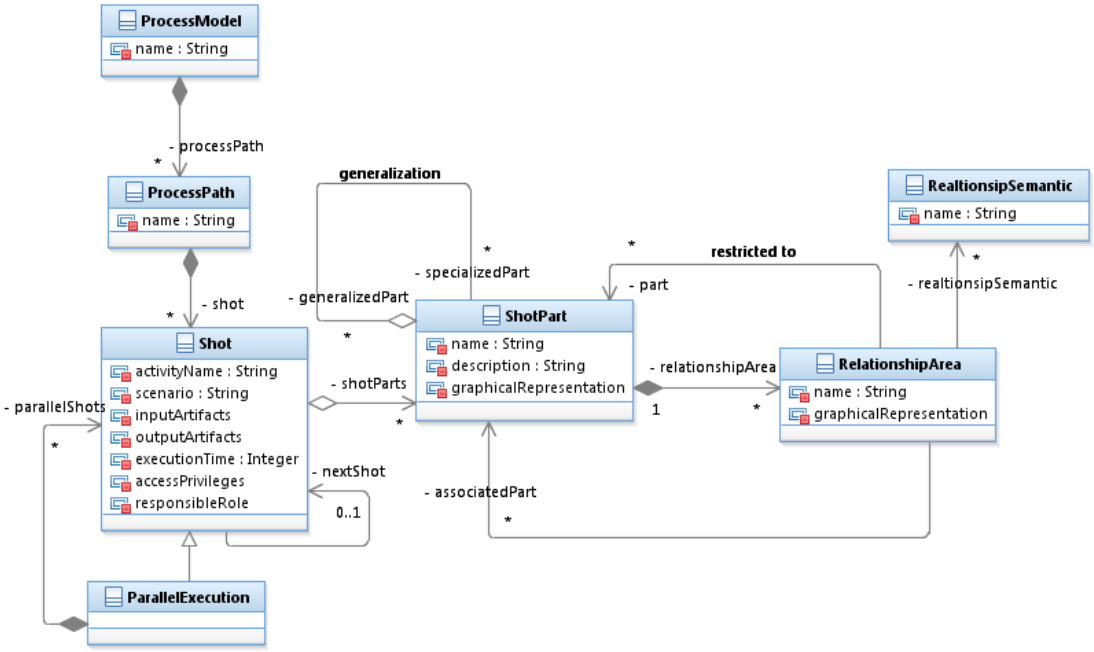
3.6 Příklad

Pokud vezme náš příklad naskladnění objednávky, můžeme tento proces zachytit následujícím obrázkem 7. Obrázek má pozadí představující sklad. V přední části budou vytvořeny aktivní oblasti pro zodpovědně pracovníky a vstupní artefakty (objednávka, krabice). Výstupní oblasti skladu budou obsahovat výstupní artefakty procesu (zabalený produkt, faktura k zaplacení). Pokud to shrneme, tak z jednoduchého obrázku můžeme vyčíst vstupní a výstupní artefakty a zodpovědné pracovníky, které se objeví v rámci procesu naskladnění.



Obrázek 7: Ilustrace kompletního snímku (převzato z [5])

3.7 Zachycení metody Storyboardů pomocí třídního diagramu



Obrázek 8: Třídní diagram metody Storyboardů (převzato z [5])

3.8 Závěr do úvodu Storyboardů

Kapitola úvod do Storyboardů měla přiblížit čtenáři alternativní techniku pro modelování podnikových procesů. Technika Storyboardů je teprve na svém začátku, ale již se ukazuje, že toto modelování je jednodušší a pro všechny zainteresované osoby v podniku přehlednější. Velký potenciál je v možnosti generovat informace pro následnou simulaci podnikových procesů.

V další části této diplomové práce se budu zabývat vývojem grafického editoru pro podporu tvorby Storyboardů.

4 Analýza

4.1 Specifikace požadavků

Na začátku celého vývoje je nutné vždy specifikovat požadavky, co má systém dělat. Tyto požadavky získáváme od zákazníků a všech zainteresovaných osob. V mém případě byl zákazníkem vedoucí mé diplomové práce Ing. David Ježek, Ph.D. Při první konzultaci jsem získal obecný přehled požadavků na vyvíjený systém.

- Vytváření ikon a jejich popis (samotný obrázek bude importován ve formátu SVG).
- Vytváření aktivních oblastí v jednotlivých ikonách a popisu příslušných vazeb, které mohou být realizovány pomocí této aktivní oblasti.
- Definici vazeb mezi jednotlivými ikonami (dědičnost, propojení, agregace).
- Skládání Storyboardů a jednotlivých záběrů z předdefinovaných ikon.
- Export vytvořených Storyboardů.

4.1.1 Knihovna ikon

Jedním z hlavních požadavků na systém, byla možnost vytvoření knihovny ikon, která se bude skládat z jednotlivých ikon. Takto vzniklé knihovny bude možné připojit k projektu. Ikony budou využívat SVG formát pro zobrazení grafického obrazu. Výhodou SVG formátu je, že popisuje výsledný obrázek pomocí XML. Do daného XML souboru je možné vložit vlastní údaje, které umožní rozšířit informaci ikony o další významné informace. Tyto informace pak bude možné následně zpracovávat a vytvářet z nich simulační modely. Knihovny ikon bude možné ukládat na disk.

4.1.2 Aktivní oblasti

Dalším z požadavků byla možnost vytváření aktivních oblastí v ikonách. Aktivní oblasti umožňují ikoně připojit další ikonu a tímto rozšířit ikonu o další významnou informaci. Aktivní oblast bude mít také omezení, které určí, jaké ikony lze připojit do dané aktivní oblasti. Představme si jednoduchý příklad, kde máme ikonu představující personálního úředníka, který má definovanou aktivní oblast ruku, která může přijmout ikonu životopis. Pokud bychom chtěli do aktivní oblasti připojit ikonu tiskárna, program takovéto spojení nedovolí. U aktivní oblasti byl požadavek na uchování názvu dané oblasti a její činnosti, kterou může oblast provádět.

4.1.3 Činnost

Činnost definuje aktivní oblasti akci, kterou může provádět. Příkladem může být akce: skartuje, odevzdává, zpracovává.

4.1.4 Skládání Storyboardů

Pokud máme vytvořené ikony, je možné z nich sestavit konkrétní snímek, který umožní zachytit proces. Z takto vytvořených snímku lze skládat jednotlivé Storyboardy.

4.1.5 Export

Dalším z požadavků byla možnost exportovat jednotlivé snímky do obrázku, které můžou být následně používány při porozumění daného procesu ve firmě.

4.2 Dokumentace požadavků

Pro sepsání požadavků jsem vybral nejjednodušší formu zápisu, prostým textem. Požadavky se organizují do skupin základních funkcí (features). K nim se dále specifikují podrobné požadavky (request) na systém.

Request (Požadavek) - je schopnost, kterou produkt musí poskytovat nebo něco, co musí produkt dělat, aby uspokojil zákaznickou potřebu.

Features (Vlastnost) - je množina společných požadavků, které umožňují uživateli uspokojit bysnys cíl nebo potřebu.
Pro editor Storyboardů byly požadovány tyto vlastností

- FEAT 1 Vytváření ikon a jejich popisu (samotný obrázek bude importován ve formátu SVG).
- FEAT 2 Vytváření aktivních oblastí v jednotlivých ikonách a popisu příslušných vazeb, které mohou být realizovány pomocí této aktivní oblasti.
- FEAT 3 Definici vazeb mezi jednotlivými ikonami (dědičnost, propojení, agregace).
- FEAT 4 Skládání Storyboardů a jednotlivých záběrů z předdefinovaných ikon.
- FEAT 5 Export vytvořených Storyboardů.

Z těchto vlastností byly vytvořeny kompletní požadavky, které jsou přiloženy v příloze.

4.3 Použité technologie

Na začátku celého vývoje jsem nejdříve definoval technologie, které byly použity při tvorbě projektu.

4.3.1 Java 7

Jelikož v zadání nebyla specifikovaná platforma, na které se měl projekt vytvářet, zvolil jsem platformu Javu. V době, kdy byla aplikace vyvíjena, již byla k dispozici verze Javy 7, která přinesla řadu novinek, které byly využity i v projektu.

4.3.1.1 Operátor diamant

Příchodem verze Javy 1.5 byly velkou novinkou tzv. Generické typy, které dovolovaly například omezit kolekci na určitý typ, který lze vložit do dané kolekce. Operátor diamant je spíše kosmetickou úpravou Javy a dovoluje nám zkrátit inicializaci kolekcí.

```
public class OperatorDiamant {

    public static void main(String[] args) {
        List<Osoba> osoby = new ArrayList<Osoba>(); // Verze Java 6
        List<Osoba> osobyVerze7 = new ArrayList<>(); // Verze Java 7
    }
}
```

Výpis 1: Operátor diamant

4.3.1.2 Switch a String

Další kosmetickou změnou přibyla možnost použít String ve výrazu switch. Switch je přepínač, neboli příkaz pro mnohonásobné větvení programu.

```
public class Switch7 {

    public static void main(String[] args) {
        String podminka = args[0];
        switch(podminka) {
            case "vypni": {
                vypniSvetlo();
            }
            case "zapni": {
                zapniSvetlo();
            }
        }
    }
}
```

Výpis 2: Výpis programu: String ve výrazu switch

4.3.2 Batik

Jelikož byl jedním z požadavků "import ikon v svg formátu", bylo nutné se poohlédnout po nějaké knihovně, která umí pracovat s svg formátem. Pro Javu existuje knihovna Batik, která byla dostačující pro vývoj aplikace.

4.3.2.1 Komponenta JSVGCanvas

Batik poskytuje pro vykreslení svg formátu, grafickou komponentu JSVGCanvas, která je kompatibilní s knihovnou swing, jež byla použita pro vývoj aplikace.

```
public class BatikCanvas {

    public static void main(String[] args) {
        String uriKSouboru = "D:/svgikony/panacek.svg";
        File ikona = new File(uriKSouboru);
```

```
            JSVGCanvas canvas = new JSVGCanvas();
            canvas.setURI(ikona.toURI().toString());
            JFrame okno = new JFrame();
            okno.getContentPane().add(canvas);
            okno.setVisible(true);
        }
    }
```

Výpis 3: Výpis programu: Použití komponenty JSVGCanvas

4.3.3 MigLayout

Tato komponenta slouží pro rozložení grafických komponent na panelu. Java implicitně umí poskytnout různé správce rozvržení pro grafické komponenty. Bohužel jsou tito správci těžkopádní, nepříjemní a na složitější rozvržení téměř nepoužitelní. Při hledání nějaké alternativy jsem narazil na knihovnu MigLayout, která se výborně hodí na práci s grafickými prvky.

4.3.4 Netbeans

Pro vývoj aplikace v Javě existují různá vývojová prostředí. Z těch hlavních to jsou Netbeans, Eclipse, IntelliJIDEA. Já si zvolil Netbeans, jelikož s ním mám nejvíce zkušeností. Konkrétně byla využita nejnovější verze 7.1

4.3.5 SVN

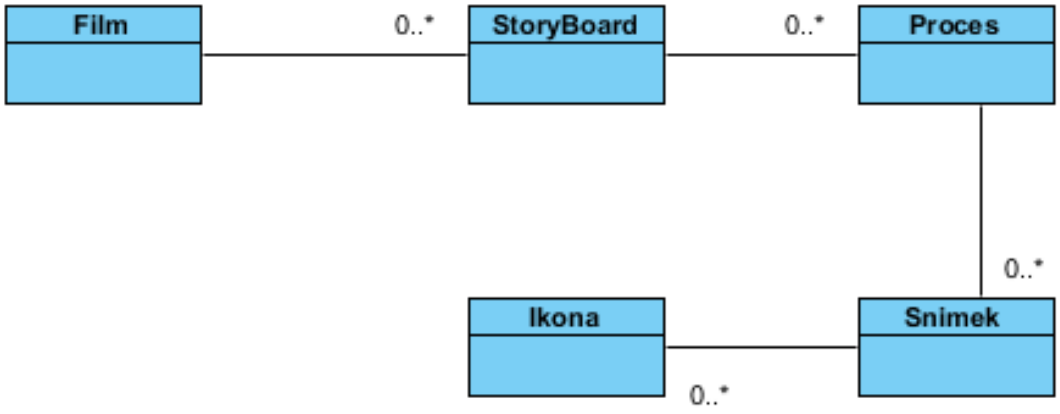
V dnešní době klíčovým prvkem při vývoji projektu je i samotná správa zdrojových kódů. Já využil verzovacího systému pro správu zdrojových kódů, konkrétně SVN. Hlavní přínosy verzovacícho systému jsou

- Zálohovatelnost
- Vrácení na předcházející verzi
- Bezpečnost
- Možnost pracovat v týmu

5 Návrh

5.1 Architektura - složení projektu

Struktura projektu částečně vychází z popisu, který je uveden v první kapitole.



Obrázek 9: Analytický model projektu

Z analytického modelu je patrné, že jeden projekt se skládá ze stromové struktury. Film může mít více Storyboardů, Storyboard může mít více procesů. Proces se skládá z jednotlivých snímků, snímek je složen z několika ikon. Analytický model umožňuje zachycení základních objektů, které se budou v architektuře objevovat. Při vytváření architektury lze vycházet z analytického modelu, jen je nutné detailněji specifikovat podrobnosti objektů, jejich vlastností a vztahy mezi nimi. Míra použitého detailu závisí na architektovi, který sám určí, jak podrobně specifikuje architekturu.

5.1.1 Rozhraní PrvekProjektu

PrvekProjektu je rozhraní, které představuje jakýkoliv objekt ve stromové struktuře, proces, snímek, Storyboard, film. Rozhraní také poskytuje řadu metod pro práci s danými objekty. Důležitou metodou je *visit(aVisit: PrvekProjektu)*, která je součástí návrhového vzoru **Visitor**, který byl implementován pro procházení hierarchické struktury. O návrhovém vzoru visitor bude pojednáno v další části textu.

5.1.2 Třída AbstraktniPrvekProjektu

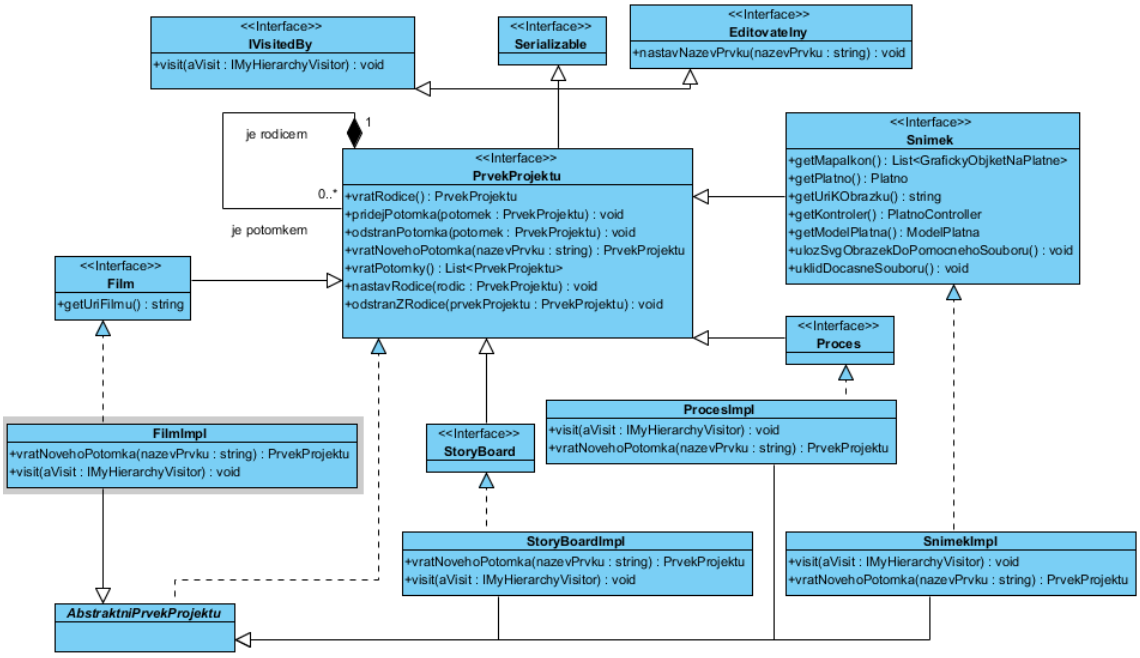
Je to abstraktní třída, která implementuje rozhraní *PrvekProjektu*. Tato třída poskytuje výchozí implementaci rozhraní *PrvekProjektu*. Tato technika je hojně využívána, neboť nám poskytuje výchozí implementaci rozhraní. Pokud se rozhraní v budoucnu změní, bude stačit implementovat tyto změny na jednom místě, aniž by bylo nutné další třídy implementující toto rozhraní měnit.

5.1.3 Rozhraní - Film, Storyboard, Proces, Snimek

Rozšiřující rozhraní, které dědí od rozhraní *PrvekProjektu* a přidává určité speciální metody, které se liší mezi objekty, a proto nebyly specifikovány v rozhraní *PrvekProjektu*.

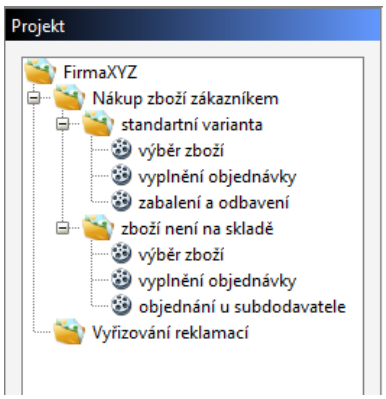
5.1.4 Třídy - FilmImpl, StoryboardImpl, ProcesImpl, SnimekImpl

Třídy dědí z abstraktní třídy *AbstraktniPrvekProjektu*, přičemž získávají výchozí implementaci rozhraní *PrvekProjektu* a také implementují svá rozšiřující rozhraní, která upřesní chování objektů.



Obrázek 10: Architektura projektu

V aplikaci pak bude jeden projekt vypadat následovně



Obrázek 11: Ukázka strukury projektu v aplikaci.

5.1.5 Rozhraní Snímek

Velice důležitým prvkem v architektuře je třída *Snímek*, která představuje byznys proces. Z teorie Storyboardů uvedené v první části se blíží snímek k definici třídy Shot (Ilustrační obrázek). Obsahuje odkaz na objekt třídy *Platno*. *Platno* uchová informace o ikonách, které jsou ke snímku připojeny, také tyto ikony vykresluje.

5.1.6 Třída Platno

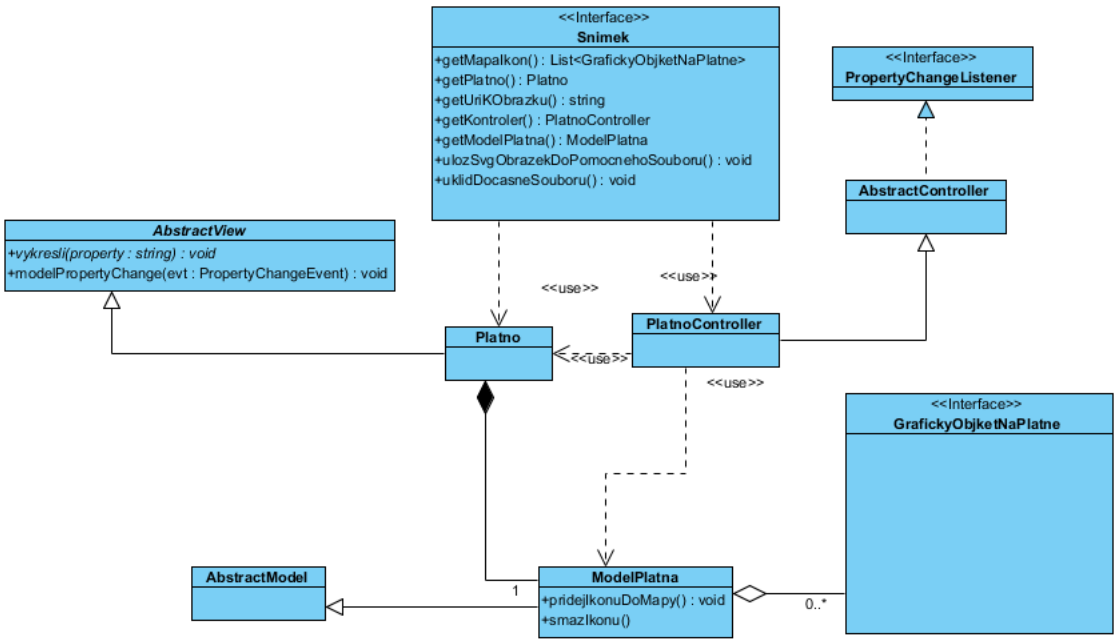
Tato třída uchovává informace o ikonách, které jsou ke snímku připojeny a také tyto ikony vykresluje. Plátno je implementováno pomocí návrhového vzoru MVC (Model-View-Controler).

5.1.7 Rozhraní GrafickýObjektNaPlatne

Toto rozhraní představuje ikonu, která může být připojena ke snímku. Rozhraní má důležitou metodu *JComponent vratGrafickouReprezentaci()*, vracející grafickou reprezentaci ikony. Pro zobrazení SVG formátu byla využita knihovna Batik a její komponenta *JSVG-Canvas*.

5.1.8 Rozhraní IAktivniOblastPrijmajici

Rozhraní definuje sadu metod, které musí být implementovány, aby objekt mohl představovat přijímací oblast. Tento objekt pak může přijmout objekt typu *GrafickýObjektNaPlatne*, takto lze pak jednotlivé ikony skládat s jinými ikonami, a tím umožňuje definovat různé vazby.



Obrázek 12: Detail snímku

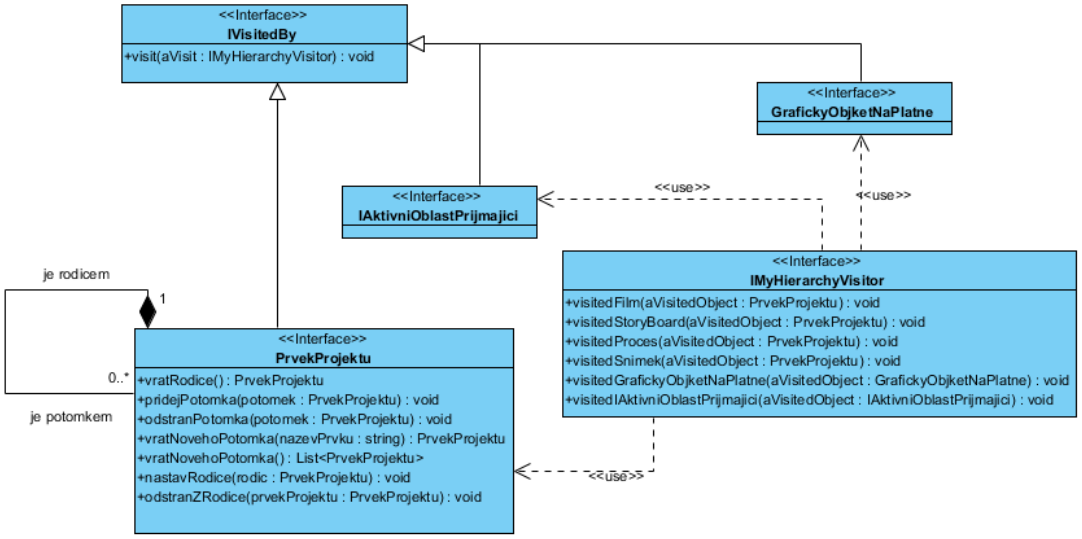
5.1.9 Návrhový vzor Visitor

Z architektury je patrné, že poskládané Storyboardy tvoří jistou hierarchii objektů. Pro procházení stromové struktury je ideální implementovat návrhový vzor visitor, který elegantně projde celou stromovou strukturu objektů. Při procházení můžeme následně získávat informace o objektech.

Z přiloženého obrázku 13 je vidět, že stačí na všech objektech, kde chceme implementovat visitora, vytvořit metodu *visit(aVisitor: IMyHierarchyVisitor)*. Podívejme se na jednoduchý příklad 4 implementace ve třídě *Snimek*.

```
public class SnimekImpl extends AbstraktniPrvekProjektu implements Snimek {

    @Override
    public void visit (IMyHierarchyVisitor aVisitor) {
        aVisitor.visitedSnimek(this);
        for(GrafickyObjketNaPlatne g: getMapaIkon()) {
            g.visit(aVisitor);
        }
    }
}
```

Výpis 4: Výpis programu: implementace visitoru ve třídě Snímek



Obrázek 13: Implementace návrhového vzoru Visitor

5.1.10 Použití Visitoru - export do xml

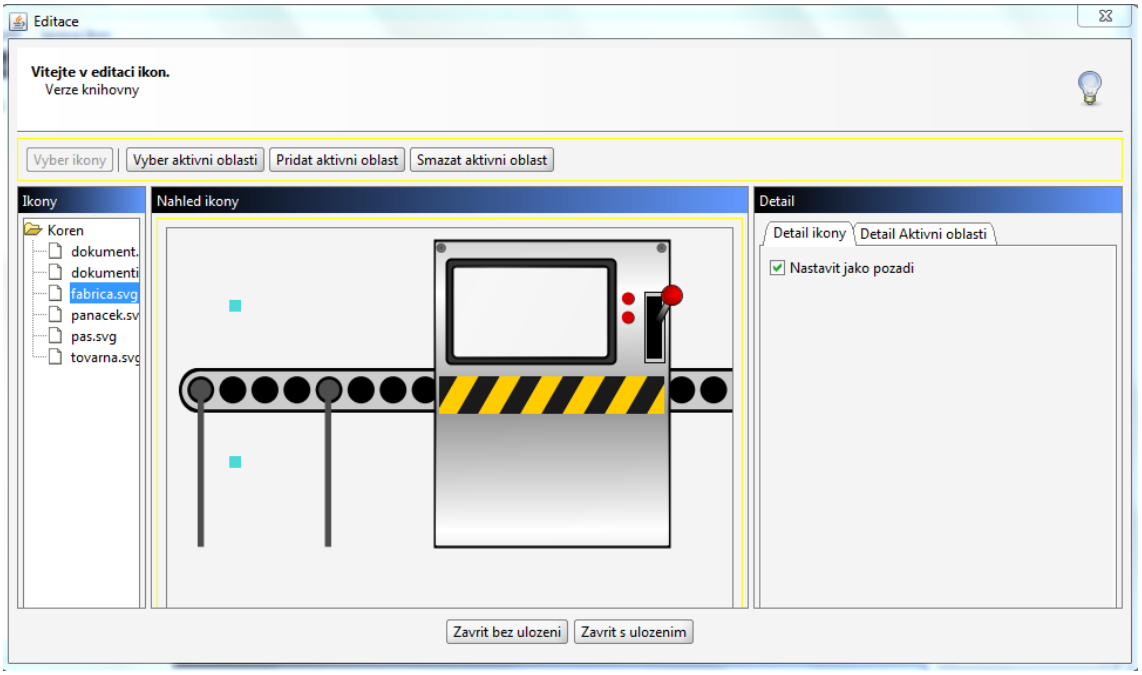
Návrhový vzor visitor jsem využil při exportu informací o objektech do xml souboru. Stačí implementovat rozhraní *IMyHierarchyVisitor*.

```
public class ExportXmlVisitor implements IMyHierarchyVisitor {
    private Element snimek,ikona,oblast;

    public Element getSnimek() {
        return snimek;
    }

    @Override
    public void visitedGrafickyObjketNaPlatne(GrafickyObjketNaPlatne aVisitedObject) {
        ikona = new Element(ElementDefinice.ELEMENT_IKONA);
        Attribute nazevIkony = new Attribute(ElementDefinice.ATRIBUT_NAZEV_IKONY,
                aVisitedObject.getNazevIkony());
        ikona.addAttribute(nazevIkony);
        snimek.appendChild(ikona);
    }

    @Override
    public void visitedIAktivniOblastPrijmajici ( IAktivniOblastPrijmajici aVisitedObject) {
        ... // sestrojeni Xml elementu z atributů objektu aVisitedObject
    }

    // zbytek implementace
}
```

Výpis 5: Výpis programu: použití visitoru - export do xml

5.2 Aplikace pro knihovnu ikon

Aplikace pro podporu Storyboardů obsahuje také druhou důležitou funkčnost, a to možnost vytvoření knihovny ikon, která se bude skládat z grafických objektů. Tyto ikony lze rozšiřovat o další informace potřebné k vytváření smysluplných Storyboardů, zachycující byznys procesy. Knihovna ikon je serializovatelná neboli perzistentně uchovatelná na úložišti. Tuto knihovnu ikon můžeme mezi projekty sdílet, což je velice zajímavá vlastnost. Můžeme celý projekt poslat kolegovi, který se může také seznámit s daným projektem. Celou aplikaci na správu knihovny můžeme vidět na následujícím obrázku 14.



Obrázek 14: Editace knihovny ikon

5.2.1 Serializace Ikony

Než se pustíme do popisování jednotlivých tříd, rozhraní aplikace pro tvorby knihovny ikon, pozastavme se nad jedním problémem, který se vyskytnul při vývoji. Problém je s grafickou komponentou *JSVGCanvas*. Tato komponenta dokáže zobrazit SVG formát, bohužel má jednu zásadní chybu, a to že není serializovatelná, konkrétně problém dělá atribut *svgDocument* typu *SVGDocument*. Pokud by uživatel chtěl uložit svůj projekt na diskové úložiště, tak by měl problém, neboť při uložení dojde k chybě. Řešení, jak ukládat ikonu, bylo vcelku jednoduché a elegantní, místo toho, aby se ukládala komponenta *JSVGCanvas*, bude se ukládat pouze textová reprezentace SVG formátu. Při načtení nebo vytvoření ikony se z dané textové reprezentace sestrojí komponenta JSVGCanvas, která

bude graficky zobrazovat požadovanou ikonu. Celou naši strategii můžeme shrnout do několika bodů:

- Získání z atributu *svgDocument* textovou reprezentaci.
- Uchovávat textovou reprezentaci SVG dokumentu.
- Při serializaci uložit textovou reprezentaci svgDocument dokumentu.
- Při deserializaci sestrojit svgDocument dokument z textové reprezentace.
- Vytvoření JSVGCanvasu a nastavit mu atribut svgDocument.

Při hledání způsobu, jak daný problém vyřešit, jsem narážel na pojmy: **Proxy, SerializationProxy, Effective Java**. Následně jsem se seznámil s knihou Effective $Java^{TM}$ $SecondEdition$ od Joshua Bloch [3, str. 312]. V této knize je kapitola, tam se zabývá serializací neserializovatelných objektů. Řešením se ukázala soukromá statická třída nazvána SerializationProxy, která uchovává stav objektů na základě námi nadefinovanými atributy. Má jednu základní metodu *readResolve*, která je volána při deserializaci, tedy načítání. My v této metodě vytvoříme nový objekt z našich uložených atributů a vrátíme tento objekt pomocí návratové hodnoty. Pro opačný postup, tedy uložení objektu, se pouze u ukládaného objektu vytvoří metoda *writeReplace*, která je volána při serializaci v této metodě se pomocí proxy vytvoří serializovatelný objekt, tento objekt se vrátí pro další pokračovaní serializace. Toto další pokračování řeší již samotná Java. Celý tento proces je znázorněn na níže uvedeném příkladě 6.

```
public class NakreslenaIkona extends JPanel implements Serializable, IIkona {
    public transient JSVGCanvas ikona;
    protected SerializovatelnyDokument dokumentSvg;
    protected transient SVGDocument svgDocument;
    protected final UUID jedinecnyIdentifikator;
    private String nazevIkony;

    public NakreslenaIkona(String stringovaReprezentaceSVGDokumentu,String nazev,
            UUID jedinecnyIdentifikator) {
             ... // incializace promennych a dalsi potrebne operace
    }

     ... // dalsi metody

    private Object writeReplace() {
        return new NakreslenaIkona.SerializationProxy(this);
    }

    private static class SerializationProxy<E> implements Serializable {
        private final String stringovaReprezentaceSVGDokumentu,nazevIkony;
        private final UUID jedinecnyIdentifikator;
        public SerializationProxy(NakreslenaIkona ikona) {
            stringovaReprezentaceSVGDokumentu = ikona.getStringovaReprezentaceSVG();
            this.nazevIkony = ikona.nazevIkony;
            this. jedinecnyIdentifikator = ikona. jedinecnyIdentifikator ;
```

```
            }

            private Object readResolve() {
                return new NakreslenaIkona(stringovaReprezentaceSVGDokumentu,
                        nazevIkony,jedinecnyIdentifikator);
            }
        }

    }
```

Výpis 6: Výpis programu: Serializační proxy

5.2.2 Třída Činnost

Třída zachycuje jednotlivou akci ikony, která je prováděna v aktivní oblasti, viz úvod do Storyboardů 3.5.

5.2.3 Třída PovolenaIkona

Každá aktivní oblast může definovat omezení týkající se možnosti vložení ikony do dané aktivní oblasti. Můžeme si představit situaci, kde máme ikonu skladník, s aktivní oblasti na ruce, bylo by špatné, kdyby šlo vložit do dané oblasti třeba auto. Proto jsem vytvořil třídu *PovolenaIkona*, která umožní nadefinovat toto omezení. Třída *PovolenaIkona* má konstruktor s třemi parametry, ten nejdůležitější představuje objekt třídy *IkonaUpravitelneOblasti*, tento objekt představuje ikonu, kterou lze vložit do aktivní oblasti.

5.2.4 Třída NakreslenaIkona

Základní třída, která pracuje s SVG formátem, implementuje jednoduché rozhraní *IIkona*, toto rozhraní obsahuje pouze jedinou metodu *UUID getJedinecnyIdentifikator()*. Tímto způsobem je zajištěno, že každá ikona má jednoznačný identifikátor, podle kterého lze určit identitu ikony. Třída *NakreslenaIkona* definuje konstruktor se třemi parametry. Jeden z parametrů je textová reprezentace SVG dokumentu. Druhým parametrem je jedinečný identifikátor, posledním parametrem je název samotné ikony. Důležitý atribut *JSVGCanvas ikona* je nastaven na **transient**, to znamená, že při serializaci a deserializaci nebude tento atribut **používán**. Objekt třídy *NakreslenaIkona* využívá služeb objektu třídy *SerializovatelnyDokument*. Objekt třídy *SerializovatelnyDokument* dokáže vytvořit atribut *svgDocument* ze stringu představující SVG formát. Pro načítání a ukládaní je k dispozici **Proxy**. Od této třídy jsou zděděny další třídy, které rozšiřují bázovou třídu o další možnosti.

5.2.5 Třída NakreslenaSvgIkona

Třída přidává především nové funkce pro práci s SVG formátem. Objekt této třídy může vytvářet, editovat a mazat SVG elementy představující aktivní oblasti. O toto rozšíření se stará implementace rozhraní *SvgEditovatelny*.

5.2.6 Třída IkonaUpravitelneOblasti

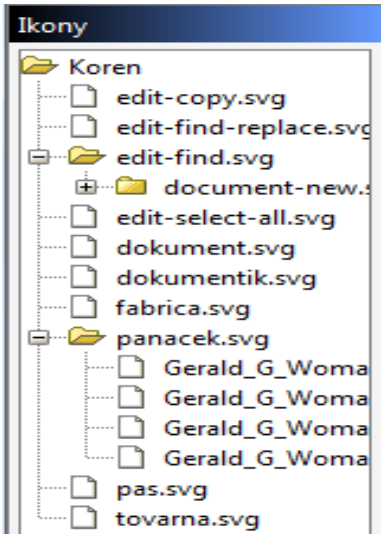
Objekty této třídy mají mnoho vlastností. Implementují rozhraní *Transferable*, které umožňuje ikoně přetáhnutí z jednoho panelu na druhý. Tento efekt využívám, když do aktivní oblasti vkládám povolené ikony, které může daná aktivní oblast přijmout. Dalším rozhraním, které přidává nové funkce, je *AktivniOblastKPozorovani*. Definuje sadu metod pro implementaci návrhového vzoru **Observer**. Při úpravě aktivních oblastí na dané ikoně potřebuji vysílat tyto změny, proto jsem využil tento jednoduchý návrhový vzor. Jde především o strukturální změny aktivních oblastí, jako je nová aktivní oblast, smazána aktivní oblast, pro tyto informace existují metody, které se volají na podnět při dané změně. Další rozhraní *IkonaKPozorovaniKliknuti* opět definuje návrhový vzor Observer. Tentokrát pro šíření události o kliknutí či nekliknutí na aktivní oblast. Tyto informace používám pro zobrazení podrobností o dané aktivní oblasti a také o její možné editaci. Pro budoucí rozšíření celé aplikace, každá ikona implementuje rozhraní *VerzeObjektu*, toto rozhraní definuje verzi objektu. Předpokládejme situaci, kdy je použita knihovna na více projektů. Pokud následně tuto knihovnu změníme, přidáme, odebereme ikony či nějak jinak pozměníme strukturu knihovny, potřebujeme tyto změny sladit s vytvořenými projekty. Toto sladění je již na další rozšíření aplikace. Dopředu jsem také implementoval poslední Observer, a to *PredmetKPozorovaniVerzeObjektu*. Stačí pouze k ikoně zaregistrovat pozorovatele, který chce dostávat informace o změnách verze na ikoně. Objekt třídy *IkonaUpravitelneOblasti* má metody na správu svých aktivních oblastí. Může tyto oblasti přidávat a mazat.

5.2.7 Třída AktivniOblastVychoziIkony

Třída definuje několik rozhraní, jednou z nich je *ISpravaPovolenychIkon*. Podle názvu rozhraní se jedná o kompletní správu povolených ikon. Dovoluje přidávat, odebírat povolené ikony aktivní oblasti, také dovoluje vracet seznam všech povolených ikon, přiřazené aktivní oblasti. Tento seznam je implementován pomocí metody *Collections.unmodifiableList(. . .)*. Tato syntaxe umožní vracet seznam, který nelze nijak měnit, pouze prohlížet kvůli bezpečnosti objektu. Rozhraní *ISpravaPovolenychIkon* pracuje s objektem třídy *ModelPovolenych*, který uchovává informace o povolených ikonách. Téma povolených ikon jsem probíral o pár odstavců výše 5.2.3, takže pro připomenutí odkážu na daný odstavec, kde se čtenář dozví o povolených ikonách a jejich souvislostech s aktivními oblastmi. Třída *AktivniOblastVychoziIkony* implementuje také rozhraní *ISpravaCinosti* z názvu je zřejmé, že se stará o správu činnosti aktivní oblasti.

5.2.8 Třída Knihovnalkon

Cílem této třídy je udržovat informace o všech ikonách v knihovně. Ikony můžou mít tendenci se organizovat do skupin, je možné mít mezi ikonami dědičnost, proto jsem zvolil reprezentaci všech ikon jako stromovou strukturu. Pro implementaci jsem využil dostupnou třídu *DefaultTreeModel*, která je součástí standardní edice Javy. Pro představu, jak může taková knihovna ikon vypadat, přikládám snímek z aplikace 15.



Obrázek 15: Náhled na knihovnu ikon a její ikony

5.2.9 Drag and drop

Předtím než se pustím do podrobností o implementaci, je třeba na začátku vysvětlit jeden důležitý pojem DnD (Drag and Drop), protože je použit v aplikaci. DnD pozná každý, kdo se již setkal s nějakou grafickou desktopovou, ale v dnešní době i internetovou aplikací. Je to funkce, která dokáže přesunout jeden objekt na jiné místo v aplikaci, klasicky pomocí myší přetáhneme objekt z jedné části aplikace do druhé. Typickým příkladem je v operačním systému přesunutí souboru do odpadkového koše. Celý tento proces můžeme rozdělit na dvě části.

5.2.9.1 Rozhraní DropTargetListener Rozhraní plní dvě důležité úlohy. Za prvé poskytuje efekty na pozadí. Za druhé obsluhuje události, k nimž dochází na straně cílového objektu. Rozhraní poskytuje celkem pět metod. Nejzajímavější metoda je *drop(DropTargetDropEvent event)*. K volání této metody dojde po každém uvolnění přesouvaného objektu nad cílovým objektem. V této metodě pak následně získáme přenášený objekt, který dále můžeme využít v rámci aplikace.

5.2.9.2 Rozhraní Transferable Pokud chceme, aby náš objekt byl posouvatelný v rámci aplikace, musí náš objekt nebo lépe řečeno naše třída implementovat rozhraní *Transferable*, které definuje sadu metod. Nejzajímavější je metoda *Object getTransferData(DataFlavor flavor)*, která sestrojí a vrátí posouvatelný objekt.

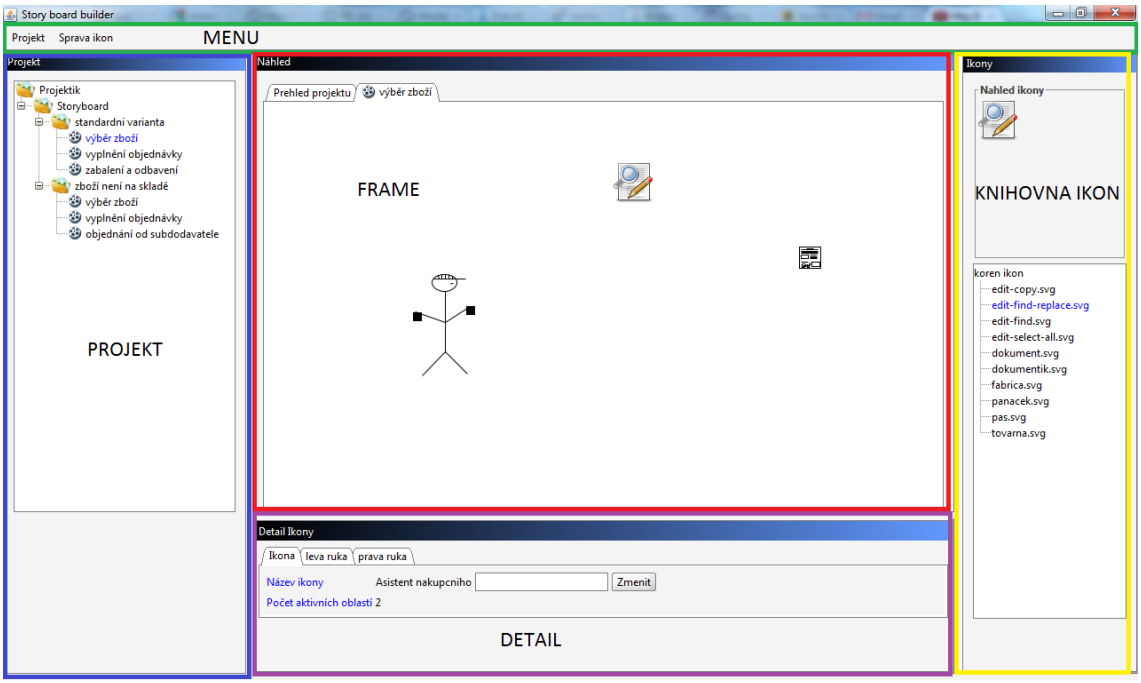
5.2.10 Správa knihovny ikon

Základní okno pro kompletní správu knihovny ikon je ve třídě *HlavniOknoEditaceIkon* v balíčku **cz.vsb.storyboardbuilder.knihovnaikon.gui**. Třída *HlavniOknoEditaceIkon* de-

finuje konstruktor s jedním parametrem, představující objekt typu *KnihovnaIkony*, tento objekt představuje samotnou knihovnu ikon, je složen především z jednotlivých ikon a pomocných metod pro správu ikon. *HlavniOknoEditaceIkon* lze rozdělit do čtyř základních oblastí představujících různé možnosti. První částí je oblast, která obsahuje tlačítka pro interakci s aktivními oblastmi na ikoně a také dovolují výběr ikony. Levá část slouží pro správu knihovny ikon, na základě objektu předaným z konstruktoru se sestrojí strom s ikonami. Pro vykreslení je využita komponenta *JTree*. Tento strom se následně nastaví pro plnohodnotné využití této komponenty. Nastaví se této komponentě vlastní editor *IkonyCellEditor*, který dokáže editovat názvy ikon pomocí dvojkliku myší. Dále je povolena vlastnost *setDragEnabled(true)*, umožní přetáhnout ikonu z jedné části aplikace do jiné. Tato vlastnost se využívá při definici povolených ikon u aktivní oblasti, kde přetažením ikony aktivní oblasti nadefinujeme omezení. Samotný proces přetažení z jedné části do druhé zajišťuje třída *DefaultMutableTreeNodeTransferHandler*, tato třída dědí od třídy *TransferHandler*, přičemž implementuje metodu *createTransferable*, která vytvoří přenositelný objekt. Komponenta *JTree* je dále nastavena na možnost přidávání ikon z počítače přes výběrový dialog. *HlavniOknoEditaceIkon* se dále skládá z panelu pro zobrazení náhledu ikony a její úpravy. *IkonaPanel* implementuje rozhraní *ObserverTlacitkaHlavniPanel*, které informuje panel s ikonou, jaké tlačítko je zmáčknuté. Na základě zvoleného tlačítka, pak aplikace vytvoří, smaže nebo edituje aktivní oblast. Posledním panelem hlavního okna pro editaci knihovny ikon je panel Sprava. Tento panel se skládá z objektů tříd *OknoStromPovolenych, SpravaCinosti, DetailAktivniOblastiPanel*, všechny třídy implementují rozhraní *ObserverAktivniOblast*, aby mohly reagovat na změnu výběru aktivní oblasti. Třída *OknoStromPovolenych* je využívána pro správu povolených ikon v aktivní oblasti, také implementuje rozhraní *DropTargetListener*, abyse objekt dané třídy stal cílem pro podporu přetažení objektů. Jednoduše řečeno, z levého stromu přetáhneme ikonu do pravého stromu, a tím definujeme povolenou ikonu, kterou může aktivní oblast přijmout. Objekt třídy *SpravaCinosti* řeší kompletní správu činnosti aktivní oblasti. Umí vkládat, editovat a mazat činnosti.

5.3 StoryBoardEditor

Druhá část aplikace slouží především pro samotné vytváření Storyboardů. Celá aplikace se skládá z pěti základních modulů. Spojení těchto modulů vytváří kompletní nástroj pro podporu metody Storyboardů. Na obrázku 16 lze nahlédnout na editor.



Obrázek 16: Storyboard editor

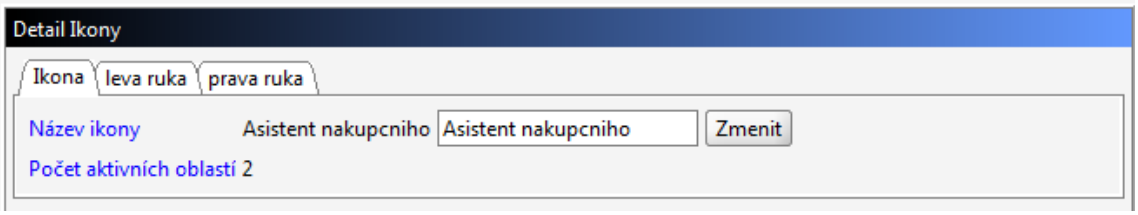
Jednotlivé barevné obdelníky rozděluji celou aplikaci na pět základních modulů:

- Modul Detail
- Modul Projekt
- Modul Knihovna ikon
- Modul Menu
- Modul Frame

V následující části si tyto moduly postupně projdeme. V každá kapitole se zmíním o důležitých třídách, rozhraních, které jsou základem architektury každého modulu.

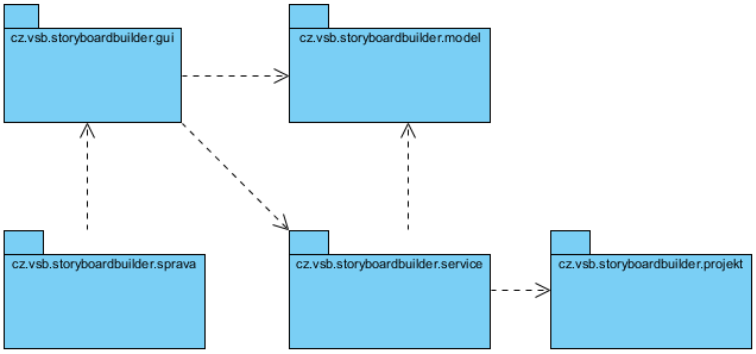
5.3.1 Modul Detail

Tento modul aplikace slouží pro zobrazování detailu projektu. V projektu je mnoho zajímavých informací, které je možné uživateli zobrazovat. Patří sem například detail jednotlivých vazeb na snímku. Detail samotné ikony, uživatel se může podívat na jednotlivé aktivní oblasti, připojené k ikoně. U aktivní oblasti je možné nahlédnout na omezení, které daná aktivní oblast poskytuje.



Obrázek 17: Náhled na detail ikony

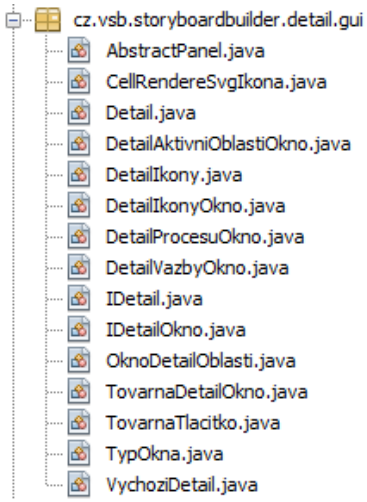
Z příkladu vidíme detail ikony s názvem Asistent nákupčího. Detail také poskytuje informaci o aktivních oblastech, daná ikona má dvě tyto oblasti. Každá záložka pak dále specifikuje detail jednotlivé aktivní oblasti. Celý modul detailu se skládá z těchto balíčků: *Gui, Sprava, Model, Service*. Na níže uvedeném obrázku 18 můžeme vidět jednotlivé závislosti mezi balíčky.



Obrázek 18: Balíčky modulu detailu

5.3.1.1 GUI Tento balíček vytváří grafické komponenty pro zobrazování detailů projektu. Základním prvkem každého detailu je rozhraní *IDetailOkno*, které definuje dvě metody, jednu pro nastavení detailu *nastavDetail(Object data)*, metoda má argument typu Object, a proto může přijmout jakýkoliv objekt, který bude reprezentovat detail. Druhá metoda rozhraní *IDetailOkno* je *JTitlePanel vratPanel()*, z názvu metody je patrné, že vrátí grafickou komponentu s detailem nějakého objektu. Konkrétní implementační třídy pak implementují rozhraní *IDetailOkno* a na základě argumentu data sestrojí grafickou komponentu, která bude reprezentovat jednotlivý detail. Balíček gui obsahuje také třídu *Tovar-*

naDetailOkno, která má jedinou statickou metodu *IDetailOkno vratDetailniOkno(TypOkna typ)*, tato metoda na základě předaného argumentu sestrojí konkrétní okno s detailem. TypOkna je výčtový typ a představuje jednotlivé detaily, například `PROCES`, `VAZBA`, `AKTIVNI_OBLAST`, `IKONA`,`VYCHOZI`. Posledním rozhraním je *IDetail*, které zobrazuje již konkrétní komponentu implementující rozhraní *IDetailOkno*. Obsahuje metodu *pripojOkno(IDetailOkno okno)*, pro připojení komponenty detailu a metodu *JComponent vratNahledDetailu()*, která vrátí náhled samotného detailu. Na přiloženém obrázku 19 se můžeme podívat na celou strukturu balíčku GUI.

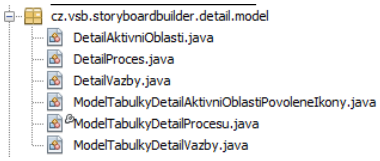


Obrázek 19: Struktura balíčku GUI

Z diagramu balíčku modelu pro detail je patrné, že balíček gui využívá balíčky model a service. Třídy z modelů představují nosiče dat jednotlivých detailů. Příkladem může být třeba model tabulky pro detail procesu. Naopak třídy z balíčku service vytvářejí data pro modely. Když jsou modely vytvořeny, grafické komponenty tyto modely zobrazí. Uživatel pak názorně vidí detaily jednotlivých prvků v projektu. Přidání nového detailu do aplikace není nijak složité, stačí definovat novou třídu implementující rozhraní *IDetailOkno* a implementovat příslušné metody. Následně se přidá do výčtového typ, nový popis detailu. Posledním krokem pro rozšíření nového detailu je upravit třídu *TovarnaDetailOkno*, kde se přidá nová podmínka, podle které se vytvoří nový detail.

5.3.1.2 Model Modely jsou jednoduchými nosiči dat o jednotlivých detailech. Komponenty pro zobrazení tyto modely využívají k získání potřebných dat pro náhled detailu.

Každý detail má svoji odpovídající třídu, jak je patrné z přiložené ukázky struktury balíčku. Některé grafické komponenty zobrazující detaily, využívají pro zobrazení tabulku, k těmto tabulkám jsou vytvořeny odpovídající třídy.

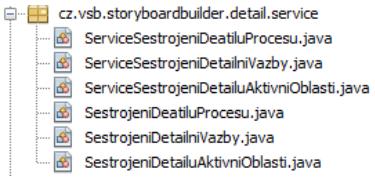


Obrázek 20: Struktura balíčku model

5.3.1.3 Service Dalším balíčkem z modulu pro detail je balíček service. Třídy a rozhraní v tomto balíčku slouží především pro vytvoření jednotlivých modelů detailu.

Opět je ke každému detailu vytvořeno rozhraní a jeho implementace. Rozhraní se většinou skládá z jedné metody nazvanou *sestrojDetail(. . .)*, návratová hodnota je vždy model konkrétního detailu, který pak grafická komponenta vykreslí.

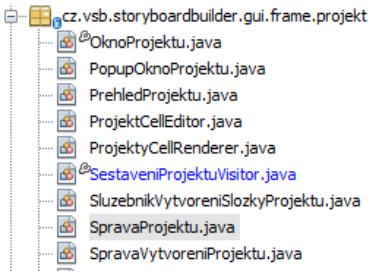
5.3.1.4 Sprava Balíček sprava, řeší hlavní koordinaci zobrazení detailu. Obsahuje rozhraní *ISpravaDetail* s metodou *zobrazDetail(TypOkna typOkna,Object data)*, která na základě parametru *typOkna* sestrojí odpovídající detail prvku v projektu. Rozhraní *ISpravaDetail* představuje spojení mezi modulem detail se zbytkem aplikace. Pokud bude chtít část aplikace pracovat s detaily, bude využívat právě zmiňované rozhraní *ISpravaDetail*.



Obrázek 21: Struktura balíčku service

5.3.2 Modul Projekt

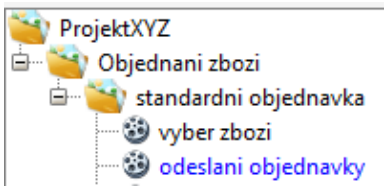
Tento modul, který najdeme v balíčku cz.vsb.storyboardbuilder.gui.frame.projekt, slouží pro kompletní správu jednoho projektu. Celou strukturu můžeme vidět na přiloženém obrázku 22.



Obrázek 22: Struktura balíčku GUI projekt

5.3.2.1 OknoProjektu Třída *OknoProjektu* představuje hlavní grafickou komponentu pro zobrazení stromové struktury projektu. Využívá komponenty *JTree* pro vykreslení hierarchické struktury projektu. Třída je také závislá na objektu třídy *HlavniOkno*, o kterém se zmíním později, pro tuto chvíli nám bude stačit, že třída *HlavniOkno* slouží pro náhled jednotlivých snímků Storyboardů. Komponenta *JTree* pro vykreslení jednotlivých uzlů používá tzv. *CellRenderer*, který má za úkol vykreslovat uzly stromu. Pro editor byl vytvořen vlastní zobrazovač, který je definován ve třídě *ProjektyCellRenderer*. Aby mohl uživatel pohodlně měnit názvy jednotlivých uzlů stromu pomocí dvojkliku myší. Vytvořil jsem vlastní CellEditor pojmenovaný *ProjektCellEditor*, objekt této třídy se nastaví na komponentě *JTree* metodou *setCellEditor(new ProjektCellEditor())*. Poslední částí nastavení komponenty *JTree* bylo vytvoření *PopupMenu*, které se zobrazí, pokud uživatel klikne na uzel stromu pravým tlačítkem myší. Podle charakteru uzlu zobrazí odpovídající nabídku možností. Toto PopupMenu je definováno ve třídě *PopupOknoProjektu*.

Na obrázku 23 lze vidět strukturu projektu, která je vykreslena v komponentě JTree. Také lze vidět vlastní zobrazovač, kde jsou využity ikony složky a pro jednotlivý snímek ikona filmového pásku.



Obrázek 23: Projekt pomocí JTree

5.3.2.2 SestaveniProjektuVisitor Na objektovém orientovaném přístupu k návrhu aplikace se mi líbí, že pokud si dáme více času na návrh jednotlivých objektů, tento čas se nám v budoucnu vrátí. Příkladem může být přidání

návrhového vzoru *visitor* do projektu.Komponenta *JTree* využívá model pro uchování stromové struktury. Při načítání projektu je tento model třeba nejdřív vytvořit. Jelikož námi implementovaný *visitor* umí projít hierarchickou strukturu projektu, využijeme tuto vlastnost pro *sestrojení* modelu pro komponentu JTree. Podívejme se na ilustrační zdrojový kód.

```
public class SestaveniProjektuVisitor implements IMyHierarchyVisitor {
    private DefaultMutableTreeNode filmNode = null;
    private DefaultMutableTreeNode storyBoardNode = null;
    private DefaultMutableTreeNode procesNode = null;
    private DefaultMutableTreeNode snimekNode = null;

    @Override
    public void visitedFilm(PrvekProjektu aVisitedObject) {
        Film film = (Film) aVisitedObject;
        filmNode = new DefaultMutableTreeNode(film);
    }

    @Override
    public void visitedStoryBoard(PrvekProjektu aVisitedObject) {
        StoryBoard s˜= (StoryBoard) aVisitedObject;
        storyBoardNode = new DefaultMutableTreeNode(s);
        filmNode.add(storyBoardNode);
    }

    @Override
    public void visitedProces(PrvekProjektu aVisitedObject) {
        Proces p = (Proces) aVisitedObject;
        procesNode = new DefaultMutableTreeNode(p);
        storyBoardNode.add(procesNode);
    }
    // v˜zájmu zpřehlednění vynechám zbytek implementace. Celý zdrojový kod je přiložen v˜
        příloze
}
```

Výpis 7: Výpis programu: Sestavení projektu pomocí návrhového vzoru visitor

Třída *SestaveniProjektuVisitor* implementuje rozhraní *IMyHierarchyVisitor*, které definuje řadu metod. Tyto metody jsou následně implementovány ve třídě *SestaveniProjektuVisitor*. Každý prvek projektu at' už to je *StoryBoard, Proces nebo Snímek*, je reprezentován v komponentě *JTree* objektem třídy *DefaultMutableTreeNode*. V ilustračním zdrojovém kódu 7 jsou nadefinovány odpovídající objekty *filmNode, storyBoardNode, procesNode, snimekNode*. Celý proces sestavení modelu projektu začíná voláním metody *visitedFilm(. . .)*.

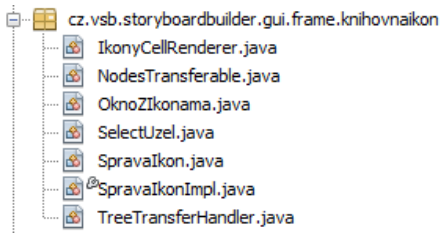
```
public void sestrojProjekt(Film film) {
        SestaveniProjektuVisitor v˜= new SestaveniProjektuVisitor();
        v.visitedFilm(film);
        film.visit(v);
        ...

}
```

Výpis 8: Výpis programu: Spuštění visitoru

V této metodě je vytvořen objekt *filmNode*, představující kořen celého stromu. Pak následuje procházení potomků objektu film a na nich volána metoda visit (. . .), kde se zavolá *visitedStoryBoard(. . .)*, kde se sestrojí uzel storyBoardNode a přidá do filmNode. Tak to pokračuje až do posledního prvku projektu, kterým je snímek. V místě, kde se spustil visitor, pak stačí zavolat metodu *getFilmNode()* třídy SestaveniProjektuVisitor, která vrátí kořen projektu, ten se vloží do modelu JTree.

5.3.3 Modul Knihovna ikon

Každý projekt je asociován s knihovnou ikon, aby mohl z jednotlivých ikon skládat Storyboardy. Pravá část editoru, jak vidíme na obrázku 16, využívá modul pro knihovnu ikon k zobrazení nadefinovaných ikon.



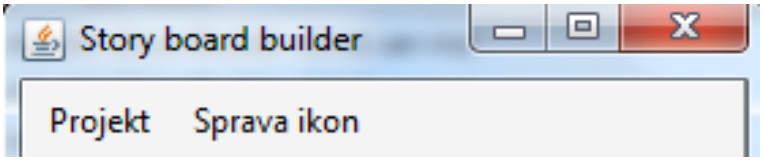
Obrázek 24: Zobrazení knihovny ikon

Strukturu tohoto modelu můžeme vidět na obrázku 24. Opět pro zobrazení stromové struktury byla využita komponenta JTree, která je ve třídě OknoZIkonama. Tato třída obsahuje také zodpovědnou část za náhled ikony. V balíčku se také nachází třída *IkonyCellRenderer* pro vlastní zobrazovač JTree. Zajímavější jsou třídy **NodesTransferable** a **TreeTransferHandler**, v kapitole 5.2.9 jsem nastínil problematiku DnD. Aby uživatel měl možnost vzít ikonu z knihovny ikon a přidat ji do snímku nebo rovnou vložit do aktivní oblasti, musí se tato funkčnost naprogramovat. K tomu, aby mohl být uzel stromu přenášen v rámci aplikace, musí se vytvořit třída, která bude implementovat rozhraní Transferable. Touto třídou je NodesTransferable, která dostane konstruktorem přenášený objekt, metodou *Object getTransferData(..)* pak tento objekt vrátí v přijímající části DnD. Druhou důležitou částí pro zprovoznění DnD je nastavení komponenty JTree, pomocí metody *setTransferHandler(..)*, která přijme objekty typu TransferHandler. K tomu slouží třída TreeTransferHandler, která dědí od TransferHandler a implementuje důležitou metodu *Transferable createTransferable(. . .)*, tato metoda vytvoří pomocí třídy NodesTransferable uzel stromu s podporou DnD. Aby mohla komponenta JTree reagovat na události výběru uzlů pomocí kliknutí myší, je vytvořena třída *SelectUzel*, která pak na základě události zobrazí náhled ikony v horní části okna.

5.3.4 Modul Menu

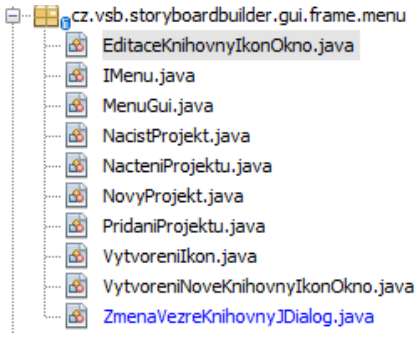
Většina grafických editorů, vývojových prostředí, aplikací obsahují také menu, které poskytuje řadu funkcí pro práci s projektem. Jde především o funkci načtení projektu, uložení projektu, zavření projektu.

Editor pro podporu Storyboardů má menu rozděleno na dvě části. Jedna část slouží pro správu projektu a druhá pro správu knihovny ikon.

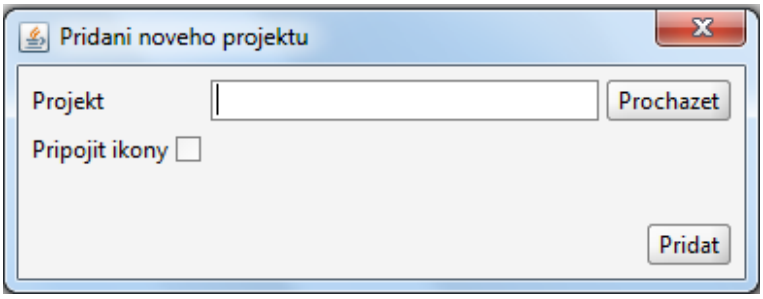


Obrázek 25: Menu aplikace

Celé menu se nachází ve třídě *MenuGui*. Java pro menu využívá komponentu *JMenu*, tato komponenta se vloží do komponenty *JMenuBar*, které vykreslí menu, jak je vidět na obrázku. Třída *MenuGui* implementuje rozhraní *IMenu*, definující jedinou metodu *JMenuBar vratMenuAplikace()*, která vrátí kompletní menu celé aplikace. Celá komponenta je složena z dalších komponent *JMenuItem*, které už představují jednotlivé funkce menu, jako je například načtení projektu, uložení projektu. Aby menu reagovalo na kliknutí myší od uživatele, je potřeba u každé komponenty *JMenuItem* zaregistrovat událost, která implementuje rozhraní *ActionListener*. Třída *MenuGui* obsahuje vnitřní třídy dědící z *AbstractAction* implementující rozhraní *ActionListener*, tyto vnitřní třídy reprezentují události jednotlivých komponent *JMenuItem*, představující funkce menu. Rozhraní *NacteniProjektu* definuje jedinou metodu *IProjekt nactiProjekt()*. Z názvu je patrné, že tato metoda má za úkol načíst projekt. Implementační třída **NacistProjekt** je zodpovědná za korektní načtení projektu, který je uložen na úložišti. Jelikož uživatel očekává jistý komfort od aplikace, obsahuje třída *NacistProjekt* grafickou komponentu *JFileChooser* pro příjemné zvolení umístění projektu. Naproti tomu rozhraní *PridaniProjektu* definuje metodu *Film vratVytvorenyNovyProjekt()*, sloužící pro vytvoření nového projektu, metoda vrací typ *Film*, který představuje počátek celé struktury projektu. Celý samotný proces vytvoření nového projektu je inicializován pomocí dialogového okna, který lze vidět na následujícím obrázku 32.

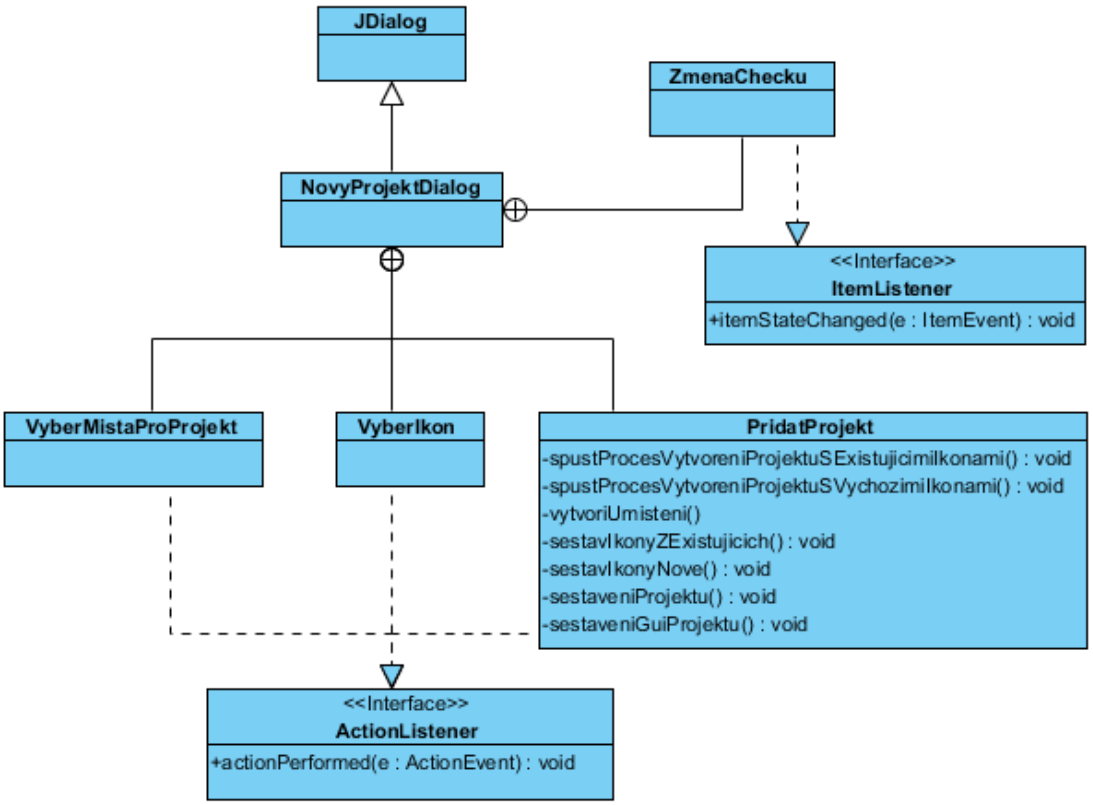


Obrázek 26: Struktura balíčku menu



Obrázek 27: Nový projekt

Toto dialogové okno se nachází ve třídě *NovyProjektDialog*. Skládá se z dalších grafických komponent, které jsou součástí Swing knihovny. Třída *NovyProjektDialog* je zodpovědná za vytvoření nového projektu, obsahuje řadu vnitřních tříd pro kompletní správu projektu. Můžeme zmínit vnitřní třídy pro vytvoření souboru představující projekt na úložišti, vnitřní třídu pro výběr umístění projektu na úložišti nebo také třídu pro výběr ikon k projektu. Celá struktura modulu menu je zachycena na níže přiloženém obrázku.



Obrázek 28: Třídní diagram pro dialogové okno - nový projekt

Nejzajímavější je třída *PridatProjekt*, která je asociována s tlačítkem **Pridat**. Na základě výběru uživatele, který zvolí možnost vytvoření knihovny ikon, aplikace spustí buď metodu *spustProcesVytvoreniProjektuSExistujicimiIkonami()*, nebo *spustProcesVytvoreniProjektuSVychozimiIkonami()*. Konkrétní proces vytvoření projektu je definován posloupností kroků spuštění metod, které korektně vytvoří nový projekt. Tyto kroky můžeme znázornit pomocí aktivitního diagramu, který je součástí specifikace UML.

5.3.5 Modul Frame

Poslední modul skládá dohromady předcházející čtyři moduly. Tento modul se nachází v balíčku: cz.vsb.storyboardbuilder.gui.frame. Tento modul sestaví celkové okno aplikace, které můžeme vidět na obrázku 16. Pro vytvoření tohoto okna jsem využil externí knihovny MultiSplit [1], která dokáže rozdělit okno na části, které lze různě rozšiřovat, zmenšovat, a tím přizpůsobovat okno na změnu velikosti zobrazované plochy. Celkové okno aplikace nalezneme ve třídě HlavniFrejmProjektu, tato třída využívá všechny probírané moduly. Využívá především hlavní okna těchto modulů, která slouží pro zobrazení informací v těchto modulech. Knihovna MultiSplit poskytuje třídu *MultiSplitPane* pro vložení a nastavení jednotlivých oken. Podívejme se na následující příklad:

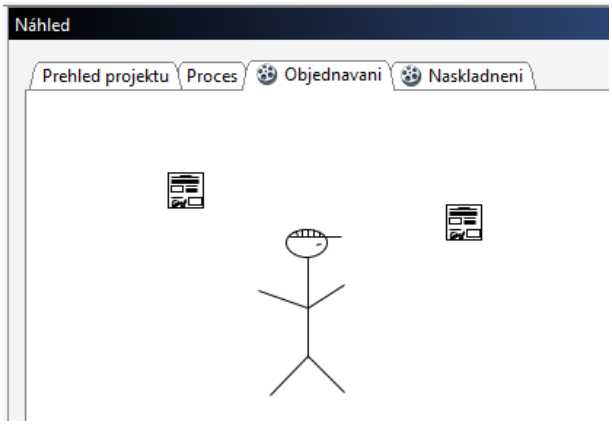
```
public void sestrojPanel() {
        String layoutDef = "(COLUMN (ROW weight=1.0 left (COLUMN middle.top middle) )
            right) bottom)";
        MultiSplitLayout.Node modelRoot = MultiSplitLayout.parseModel(layoutDef);
        multiSplitPane = new MultiSplitPane();
        multiSplitPane.getMultiSplitLayout().setModel(modelRoot);
        JTitlePanel oknoProjektuTitle = new JTitlePanel("Projekt", null,
                oknoProjektu.vratPanelSProjekty(), null);
        multiSplitPane.add(oknoProjektuTitle, " left ");
        multiSplitPane.add(oknoIkony, "right");
        JTitlePanel oknoHlavniTitle = new JTitlePanel("Náhled", null,
                hlavniOkno, null);
        multiSplitPane.add(oknoHlavniTitle, "middle.top");
        SpravceGui.getInstance().zobrazVychiziDetail();
        multiSplitPane.add(SpravceGui.getInstance().vratDetailOkno().vratNahledDetailu(), "middle"
            );
    }
```

Výpis 9: Výpis programu: Konfigurace hlavního okna

Z příkladu je patrné, že nastavení *MultiSplitPane* probíhá pomocí textové konfigurace. Pomocí třídy *MultiSplitLayout* a její metody parseModel(String layoutDef) se sestrojí model představující rozložení panelu. Tento model se pak připojí k objektu třídy *MultiSplitPane (multiSplitPane)*. Z příkladu lze ještě vyčíst, že pro jednotlivá okna modulů využívám třídy *JTitlePanel*, tato komponenta vytvoří panel s jednoduchým nadpisem s barevným pozadím, jak můžeme vidět na obrázku 16. V balíčku je také třída HlavniOkno, slouží pro zobrazování informací z projektu, ale především tato třída dokáže zobrazit jednotlivé snímky, respektive instance třídy Platno. Takto může pak uživatel přidávat do jednotlivého snímku ikony. Třída HlavniOkno obsahuje metodu zobrazSnimek(Snimek snimek), pomocí které zobrazí plátno. Plátno s ikonami se zobrazí v samostatné záložce, jak lze vidět na přiloženém obrázku 29.

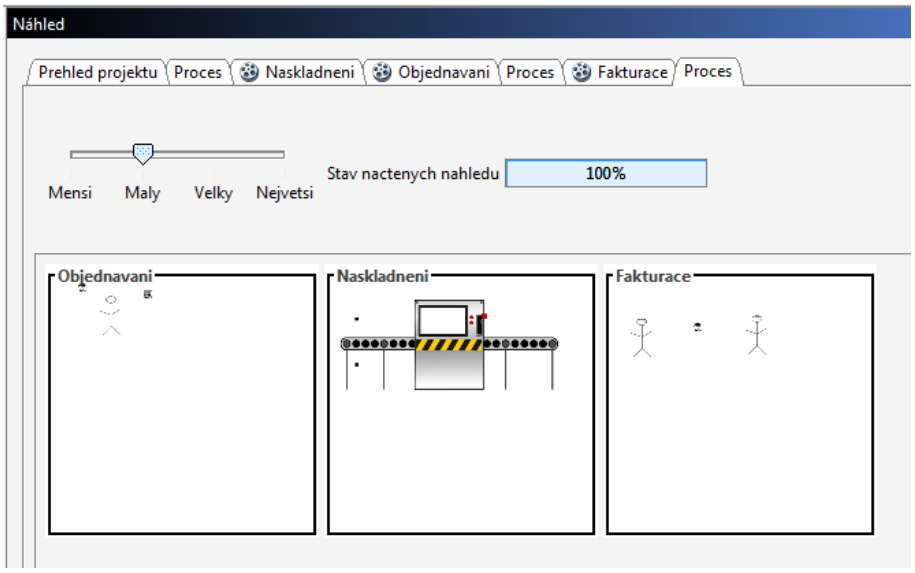
Pro příjemný přehled jednoho procesu, který se skládá z několika sekvencí snímků, umí třída HlavníOkno sestrojit detail procesu, pomocí metody zobrazProces(Proces proces). Metoda sestrojí objekt třídy NahledProcesu, který představuje grafickou kompo-

[1] Více o použití knihovny MultiSplit zde http://today.java.net/pub/a/today/2006/03/23/multi-split-pane.html



Obrázek 29: Snímek v záložce

nentu pro zobrazení sekvence snímku jednoho procesu. Příklad takové sekvence snímku můžeme vidět na následujícím obrázku 30.



Obrázek 30: Sekvence snímku v procesu

5.4 Shrnutí vývoje aplikace pro podporu metody Storyboardů

Celá aplikace se skládá z přibližně 200 modulů. V tomto případě rozumějme modulem: třídu, rozhraní, výčtový typ, proto nebylo možné popsat veškeré tyto moduly. Snažil jsem se vybrat nejzajímavější a nejdůležitější moduly architektury aplikace. Některé prvky architektury jsem doprovodil zdrojovým kódem, který měl více přiblížit daný problém. Statický pohled na určité části architektury jsem ilustroval pomocí třídních diagramů,

kde lze zachytit vzájemné vztahy mezi moduly. Také jsem upozornil v textu na využití návrhových vzorů, které umí řešit složitější konstrukce návrhu. Použití těchto návrhových vzorů ocení především budoucí programátoři, kteří budou mít za úkol rozšířit stávající aplikaci, neboť implementace návrhových vzorů lépe dovoluje rozšíření kódu, aniž by došlo k výraznému zásahu do již stávajícího kódu.

6 Závěr

Tato práce měla ukázat jistou alternativu pro modelování podnikových procesů. Metoda Storyboardů, které jsem věnoval první část, dává jiný náhled na modelování těchto procesů. Umožňuje vytvářet modely podnikových procesů na základě jednoduchých, snadno pochopitelných grafických prvků, jež pochopí každý, kdo se s touto metodou setká. Určitě stojí za zmínku říct, že metoda Storyboardů bude silnou metodou pro oblast simulací podnikových procesů, nebot' nebude zapotřebí ručně, pracně tyto simulace vytvářet, ale automaticky generovat.
Druhá část diplomové práce si kladla za cíl seznámit s návrhem a implementaci editoru pro podporu metody Storyboardů. Na začátku byly definovány jisté požadavky na systém, ze kterých následně vznikala implementace. Editor byl vytvářen ve snaze dodržovat principy objektově orientovaného programování. Pro zajímavé části kódu byly implementovány návrhové vzory, které jsou základem pro moderní objektově orientované aplikace. Editor je připraven na budoucí rozšíření. Určitě bych se zaměřil na dolování dat z jednotlivých snímků kde se tyto data budou hodit pro budoucí simulace.

Marek Garbulinský

7 Reference

[1] GRASSEOVÁ Monika a kolektiv. *Procesní řízení ve veřejném i soukromém sektoru*. Brno: Computer Press, a.s., 2008. 978-80-251-1987-7

[2] ŘEPA, Václav. *Podnikové procesy: Procesní řízení a modelování. 2., aktualizované a rozšířené vydání.*. Havlíčkův Brod: Grada Publishing, a.s., 2007. ISBN 978-80-247-2252-8.

[3] BLOCH Joshua. *Effective $Java^{TM}$ Second Edition*. New York: Addison-Wesley Pub. Co., 2008. ISBN-13: 978-0-321-35668-0, ISBN-10: 0-321-35668-3

[4] JUCHOVÁ Veronika, ŠTOLFA Svatopluk, JEŽEK David a VONDRÁK Ivo. *Storyboards in Business Process Modeling*.. In Industrial Simulation Conferences, ISC 2010

[5] JEŽEK David, ŠTOLFA Svatopluk *STORYBOARDS METHOD AND STRUCTURED SHOT*. Department of Computer Science, VSB - Technical University of Ostrava 17.listopadu 15, Ostrava-Poruba, Czech Republic

A Příloha tabulky

FEAT 1	Vytváření ikon a jejich popisu.
REQ 1.1	Uživatel má možnost importovat ikonu v svg formátu.
REQ 1.2	Uživatel může k jednotlivým ikon vložit popis.
REQ 1.3	Systém umožní uživateli vytvořit z daných ikon knihovnu.
REQ 1.4	Systém umožní ukládat knihovnu ikon do souboru.
REQ 1.5	Systém umožní načítat knihovnu ikon ze souboru.

Tabulka 2: FEAT 1

FEAT 2	Vytváření aktivních oblastí v jednotlivých ikonách a popisu příslušných vazeb, které mohou být realizovány pomocí této aktivní oblasti.
REQ 2.1	Systém umožní k ikonám vytvořit aktivní oblast.
REQ 2.2	Systém dokáže k aktivní oblasti připojit informace.
REQ 2.3	Uživatel má možnost k dané aktivní oblasti přidat informaci o činnosti, kterou daná aktivní oblast může poskytnout.
REQ 2.4	Uživatel bude mít možnost nastavit aktivní oblasti ikony, které lze připojit k dané aktivní oblasti.

Tabulka 3: FEAT 2

FEAT 4	Skládání Storyboardů a jednotlivých záběrů z předefinovaných ikon.
REQ 4.1	Systém umožní uživateli vytvářet jednotlivé snímky představující proces.
REQ 4.2	Uživatel bude moci do snímku přidávat jednotlivé ikony z knihovny ikon.
REQ 4.3	Uživatel bude moci mazat ikony ze snímku.
REQ 4.4	Systém umožní uživateli vložit k procesu/snímku pozadí.
REQ 4.5	Uživatel bude moci připojit ikonu do aktivní oblasti.

Tabulka 4: FEAT 4

FEAT 5	Export vytvořených Storyboardů.
REQ 5.1	Uživatel bude moci exportovat vytvořený snímek do obrázku.
REQ 5.2	Uživatel bude moci exportovat vytvořený snímek do textového popisu.

Tabulka 5: FEAT 5

B Příloha Zdrojové kódy

```
package cz.vsb.storyboardbuilder.gui.frame.projekt;

import cz.vsb.storyboardbuilder.ikona.snimek.GrafickyObjketNaPlatne;
import cz.vsb.storyboardbuilder.ikona.snimek. IAktivniOblastPrijmajici ;
import cz.vsb.storyboardbuilder.projekt.*;
import javax.swing.tree.DefaultMutableTreeNode;

public class SestaveniProjektuVisitor implements IMyHierarchyVisitor {
    private DefaultMutableTreeNode filmNode = null;
    private DefaultMutableTreeNode storyBoardNode = null;
    private DefaultMutableTreeNode procesNode = null;
    private DefaultMutableTreeNode snimekNode = null;

    public DefaultMutableTreeNode getFilmNode() {
        return filmNode;
    }

    @Override
    public void visitedFilm(PrvekProjektu aVisitedObject) {
        Film film = (Film) aVisitedObject;
        filmNode = new DefaultMutableTreeNode(film);
    }

    @Override
    public void visitedStoryBoard(PrvekProjektu aVisitedObject) {
        StoryBoard s˜= (StoryBoard) aVisitedObject;
        storyBoardNode = new DefaultMutableTreeNode(s);
        filmNode.add(storyBoardNode);
    }

    @Override
    public void visitedProces(PrvekProjektu aVisitedObject) {
        Proces p = (Proces) aVisitedObject;
        procesNode = new DefaultMutableTreeNode(p);
        storyBoardNode.add(procesNode);
    }

    @Override
    public void visitedSnimek(PrvekProjektu aVisitedObject) {
        Snimek sni = (Snimek) aVisitedObject;
        snimekNode = new DefaultMutableTreeNode(sni);
        procesNode.add(snimekNode);
    }

    @Override
    public void visitedGrafickyObjketNaPlatne(GrafickyObjketNaPlatne aVisitedObject) {
    }

    @Override
    public void visitedIAktivniOblastPrijmajici ( IAktivniOblastPrijmajici aVisitedObject) {
    }
```

```
}
```

Výpis 10: Výpis programu: Implementace visitoru

```
public class ExportXmlVisitor implements IMyHierarchyVisitor {
    private Element snimek,ikona,oblast;

    public Element getSnimek() {
        return snimek;
    }

    @Override
    public void visitedGrafickyObjketNaPlatne(GrafickyObjketNaPlatne aVisitedObject) {
        ikona = new Element(ElementDefinice.ELEMENT_IKONA);
        Attribute nazevIkony = new Attribute(ElementDefinice.ATRIBUT_NAZEV_IKONY,
                aVisitedObject.getNazevIkony());
        ikona.addAttribute(nazevIkony);
        snimek.appendChild(ikona);
    }

    @Override
    public void visitedIAktivniOblastPrijmajici ( IAktivniOblastPrijmajici aVisitedObject) {
        oblast = new Element(ElementDefinice.ELEMENT_AKTIVNI_OBLAST);
        Attribute a = new Attribute(ElementDefinice.ATRIBUT_NAZEV_AKTIVNI_OBLASTI,
                aVisitedObject.getNazevAktivniOblasti());
        oblast.addAttribute(a);
        if (aVisitedObject.isObsazena()) {
            Element pripojenaIkona = new Element(ElementDefinice.ATRIBUT_VLOZENA_IKONA);
            pripojenaIkona.appendChild(aVisitedObject.vratVlozenouIkonuZAktivniOblasti().
                    getNazevIkony());
            oblast.appendChild(pripojenaIkona);
            Element provadenaCinost = new Element(ElementDefinice.
                    ATRIBUT_PROVADENA_CINOST);
            provadenaCinost.appendChild(aVisitedObject.getCinost().nazevCinosti);
            oblast.appendChild(provadenaCinost);
        }
        ikona.appendChild(oblast);
    }

    @Override
    public void visitedSnimek(PrvekProjektu aVisitedObject) {
        snimek = new Element(ElementDefinice.ELEMENT_SNIMEK);
        Attribute nazevSnimkuAtr = new Attribute(ElementDefinice.ATRIBUT_NAZEV_SNIMKU,
                aVisitedObject.vratNazevPrvku());
        snimek.addAttribute(nazevSnimkuAtr);
    }

    @Override
    public void visitedFilm(PrvekProjektu aVisitedObject) {
    }

    @Override
    public void visitedStoryBoard(PrvekProjektu aVisitedObject) {
```

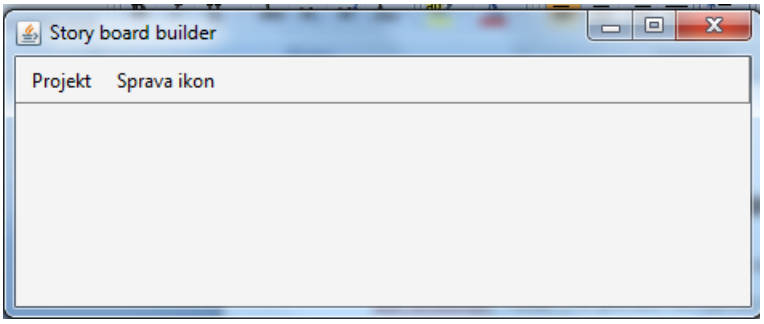
```
        }

        @Override
        public void visitedProces(PrvekProjektu aVisitedObject) {

        }

    }
```

Výpis 11: Výpis programu: Implementace visitoru

C Uživatelská příručka

C.1 Instalace a spuštění programu Storyboardbuilder

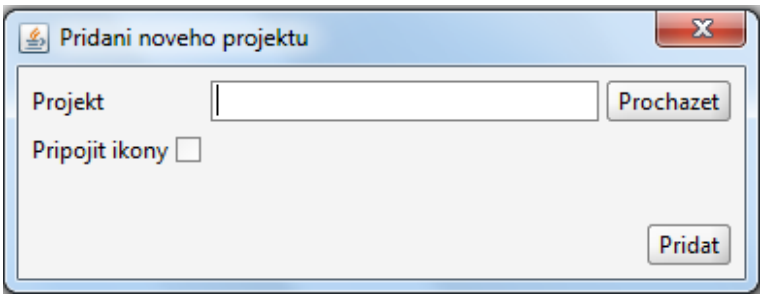
Pro korektní běh aplikace musíme mít nainstalovanou Javu, konkrétně JRE (Java Runtime Enviroment) verze 1.6, pokud ji nemáme, stáhneme z oficiálních stránek a nainstalujeme. Pokud máme Javu připravenou, spuštění aplikace můžeme provést přes příkazovou řádku *java -jar [cesta k souboru]/StoryBoardEditor.jar* nebo kliknutím myší na daný soubor. Po úspěšném spuštění se zobrazí úvodní okno.



Obrázek 31: Nový projekt

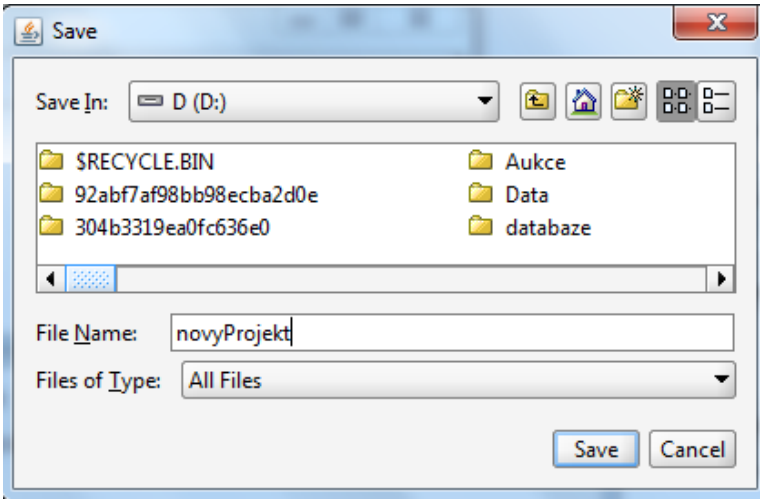
C.2 Vytvoření nového projektu

Pro vytvoření nového projektu stačí kliknout na menu Projekt a vybrat z možností Přidat. Zobrazí se nové okno pro vytvoření projektu



Obrázek 32: Nový projekt

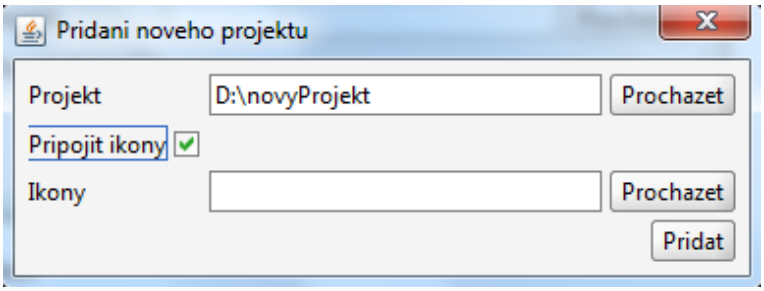
Pomocí tlačítka Prochazet, zvolíme umístění projektu, zadáme název projektu a zmáčkneme tlačítko save.



Obrázek 33: Zvolení umístění projektu

C.2.1 Zvolení knihovny ikon

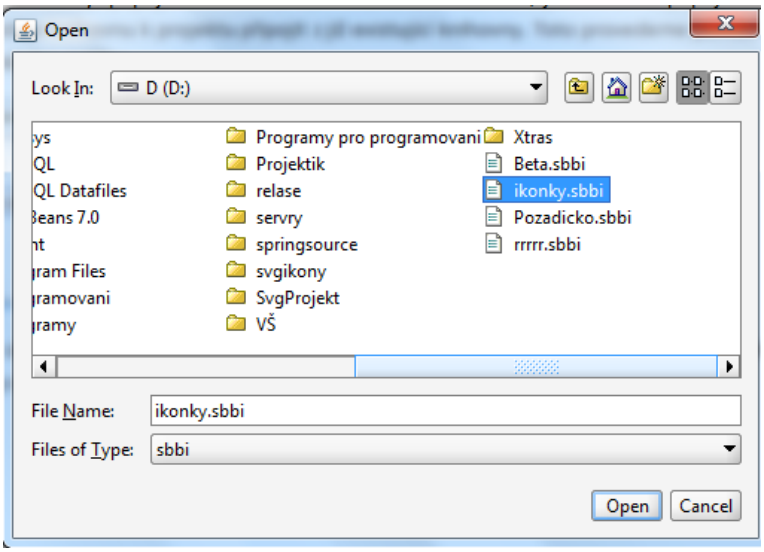
K projektu musí být připojena knihovna ikon. Máme dvě možnosti jak knihovnu připojit. Jednou z možností je knihovnu k projektu připojit z již existující knihovny. Toto provedeme pomocí zaškrtávajícího políčka.



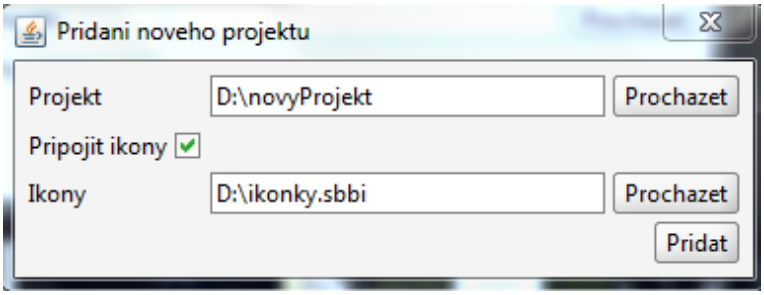
Obrázek 34: Připojení ikon

Aplikace zobrazí novou část pro připojení knihovny. Přes tlačítko Prochazet zvolíme umístění knihovny a tlačítkem open potvrdíme.

Druhou možností, jak připojit knihovnu, je vytvoření nové knihovny, tedy nezaškrtneme možnost připojení a při vytvoření projektu se ve stejné složce, kde se nachází projekt, vytvoří také knihovna ikon, kterou můžeme následně editovat. Pro dokončení vytvoření projektu již stačí potvrdit tlačítkem Pridat.



Obrázek 35: Výběr ikon



Obrázek 36: Potvrzení vytvoření projektu

D Obsah přiloženého CD

Složka Diplomka - kompletní netbenas projekt, obsahuje zdrojové kódy.
StoryBoardEditor.jar - aplikace ke spuštění